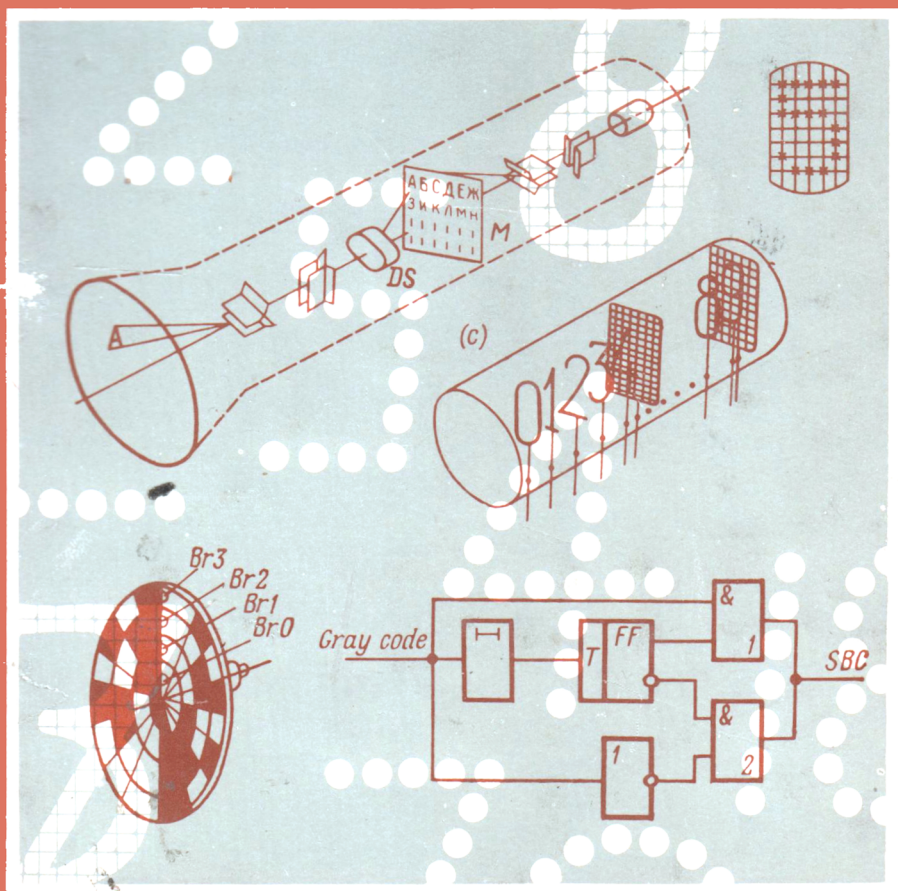


# AN INTRODUCTION TO COMPUTERS

N. Sergeev, N. Vashkevich



Mir Publishers

The late **PROFESSOR SERGEEV N.P.**, Doctor of Technical Sciences, was Head of the Chair (automatic information retrieval systems), rector of the Penza Polytechnic Institute.

Lectured on computers (both digital and analog) and engineering and business applications of computers. Read lectures on automatic information retrieval systems.

Has penned a total of 250 papers of which 200 have been published, including three monographs and 23 study aids on computer engineering and data processing.

# **AN INTRODUCTION TO COMPUTERS**

**N. Sergeev, N. Vashkevich**











**Н. П. Сергеев, Н. П. Вашкевич**

# **Основы вычислительной техники**

---

**Издательство «Вышая школа» · Москва**

N. Sergeev, N. Vashkevich

# An Introduction to Computers

*Translated from the Russian  
by  
Boris KUZNETSOV*

MIR PUBLISHERS  
MOSCOW

First published 1976  
 Revised from the 1973 Russian edition  
 Second printing 1981  
 Third printing 1987

### **The Russian Alphabet and Transliteration**

Аа	a	Кк	k	Хх	kh
Бб	b	Лл	l	Цц	ts
Вв	v	Мм	m	Чч	ch
Гг	g	Нн	n	Шш	sh
Дд	d	Оо	o	Щщ	shch
Ее	e	Пп	p	Ъ	"
Ёё	ë	Рр	r	Ы	y
Жж	zh	Сс	s	Ь	'
Зз	z	Тт	t	Ээ	e
Ии	i	Уу	u	Юю	yu
Йй	y	Фф	f	Яя	ya

### **The Greek Alphabet**

Αα	Alpha	Ιι	Iota	Ρρ	Rho
Ββ	Beta	Κκ	Kappa	Σσ	Sigma
Γγ	Gamma	Λλ	Lambda	Ττ	Tau
Δδ	Delta	Μμ	Mu	Υυ	Upsilon
Εε	Epsilon	Νν	Nu	Φφ	Phi
Ζζ	Zeta	Ξξ	Xi	Χχ	Chi
Ηη	Eta	Οο	Omicron	Ψψ	Psi
Θθ	Theta	Ππ	Pi	Ωω	Omega

*На английском языке*

© English translation, Mir Publishers, 1976.

*Printed in the Union of Soviet Socialist Republics*

# Contents

Preface . . . . .	8
Introduction . . . . .	9
<b>Part One. ANALOG COMPUTERS . . . . .</b>	<b>11</b>
<b>Chapter I. Basic Theory of Similitude and Electric Modelling . . . . .</b>	<b>11</b>
1.1. Basic Concepts of Similitude Theory . . . . .	11
1.2. Synthesis of Electric Models for Physical Systems . . . . .	18
1.3. Electric Modelling of Physical Fields . . . . .	23
<b>Chapter II. Computing Elements . . . . .</b>	<b>34</b>
2.1. General . . . . .	34
2.2. Basic Elements of Computing Circuits . . . . .	35
2.3. Summing Networks . . . . .	54
2.4. Differentiators and Integrators . . . . .	58
2.5. Function Generators . . . . .	66
2.6. Function Multipliers-Dividers . . . . .	85
<b>Chapter III. Synthesis of Analog Differential Analyzers . . . . .</b>	<b>97</b>
3.1. Development of a Block Diagram for a Differential Analyzer . . . . .	97
3.2. Main Units of Differential Analyzers . . . . .	102
3.3. Problem Preparation for a Differential Analyzer . . . . .	106
3.4. Problem Solving on an Electronic Differential Analyzer . . . . .	113
<b>Chapter IV. Algebraic and Transcendental Equation Solvers . . . . .</b>	<b>118</b>
4.1. Special-purpose Algebraic Equation Solvers . . . . .	118
4.2. Techniques for Solving Linear Algebraic Equations on Electronic Analog Computers . . . . .	123
<b>Part Two. ELECTRONIC DIGITAL COMPUTERS . . . . .</b>	<b>127</b>
<b>Chapter V. Mathematical Basis of Digital Computers . . . . .</b>	<b>127</b>
5.1. General . . . . .	127
5.2. Arithmetic Principles . . . . .	130
5.3. Switching Circuits and Boolean Algebra . . . . .	142
<b>Chapter VI. Basic Functional Elements of Electronic Digital Computers . . . . .</b>	<b>156</b>
6.1. Computer Elements . . . . .	156
6.2. Logical Design . . . . .	177
6.3. Logic Assemblies of Electronic Digital Computers . . . . .	186
<b>Chapter VII. Principal Units of an Electronic Digital Computer . . . . .</b>	<b>200</b>
7.1. Arithmetic Unit . . . . .	200
7.2. Storage . . . . .	204

7.3. Input/Output Equipment . . . . .	227
7.4. Control Equipment . . . . .	245
Chapter VIII. Principles of Programming . . . . .	252
8.1. Basic Definitions. Hypothetical Computer . . . . .	252
8.2. Direct Programming . . . . .	256
8.3. An Outline of FORTRAN Language . . . . .	269
Chapter IX. Special-purpose Computers . . . . .	287
9.1. Control Computers . . . . .	287
9.2. Digital Differential Analyzers . . . . .	289
9.3. Analog-Digital Computers . . . . .	295
9.4. Prospects in the Development and Use of Electronic Computers	296
Bibliography . . . . .	299
Index . . . . .	300



## FOREWORD

*Because of the rapid growth and wide use of computers, in practically all fields of man's activity, there is a need for students in all specialities to have a concise, yet thorough presentation of computer theory, design and operation.*

*This book is an attempt to fill this need. It covers the circuitry and operating principles of analog and digital computers, special-purpose computing devices, machines and systems. Ample space is devoted to a basic theory of similitude and simulation, and a fairly detailed description is given of basic functional elements, assemblies and units which make up typical analog and digital computers, in conjunction with an introduction to the mathematical and logical basis of electronic digital computers.*

*Separate sections deal with block-diagram synthesis of analog computers to solve algebraic, transcendental, ordinary and partial differential equations and their systems. The reader will find a survey of methods used to program problemsolving on analog and digital computers, analog with a brief outline of digital differential analyzers and hybrid systems.*

*Written by leading authorities in the field of Soviet computer engineering with a wealth of teaching and research experience at colleges, the book is student-oriented in the presentation of the subject-matter.*

*The book will be of primary value to college students and faculty members and also to researchers and engineers concerned with the development and application of computers.*

With the rapid growth and wide use of computers in practically all fields of economics, science and technology, the need has arisen to give students in all specialties a concise, yet thorough source of information on the theory, design and operation of computing devices and machines.

This book is an attempt to fill this need. It is in two parts. Part One, devoted to analog computers, sets forth the basic principles of the theory of similitude and modelling, describes the various computing elements, and introduces the reader to logic design and problem preparation. Part Two, set aside for digital computers, outlines number systems, arithmetic and logical operations, basic Boolean algebra and the theory of switching circuits. Ample space is devoted to a description of computer elements and units, programming and problem solving on a digital computer. In presenting the material, the authors have based themselves on the circuits and elements actually used in present-day Soviet computers.

The Introduction, Chapters II, IV and IX, and Sec. 7.3 of Chapter VII have been contributed by N. P. Sergeev; Chapters V, VI, VII, and VIII by N. P. Vashkevich; Chapters I and III jointly by N. P. Sergeev and Ye. A. Alexandrova; Sec. 7.2 of Chapter VII jointly by V. N. Sorokin and N. P. Vashkevich; Secs. 7.4, 8.1, 8.2 and 8.3 jointly by Yu. A. Silvestruk and N. P. Vashkevich.

The authors wish to express their deep gratitude to the faculty members of the Chair of Mathematical Machines at the Baumann Higher Technical School, and also to Professor B. V. Anisimov, Dr. Tech. Sc., and Professor E. I. Gitis, Dr. Tech. Sc., who reviewed the manuscript, for valuable suggestions and ideas which have been incorporated in the book and improved the presentation.

## Introduction

To-day, computers are active in almost all divisions of science and technology, and are doing their jobs efficiently. Apart from pure computational work, they control production processes, run traffic, handle statistical work, do economic planning, gather and process information, and solve logic and other problems.

All computers may be divided into two broad classes, analog and digital. In turn, each of these two classes may further be subdivided into general-purpose machines capable of tackling a wide range of mathematical problems, and special-purpose machines dedicated to a narrow class of problems or even a specific problem. Although less versatile because of the more regimented relations between their units, special-purpose computers are more reliable, use control, arithmetic, storage and other units of a simpler structure, and are compact. The class of special-purpose computers includes, for example, control computers, digital differential analyzers, and a variety of hybrid computing systems. Recent years have seen a marked increase in the use of hybrid computers, that is, machines combining analog and digital principles.

*Analog computers* operate on mathematical variables which appear as continuously varying physical quantities. The basis of analog computers is modelling, when a real physical process (or plant) or its elements are replaced with a model having the same properties. With modelling, an investigation can be carried out in a simpler way, conveniently and inexpensively. Of all existing techniques of modelling, widest use has to date been made of physical modelling and mathematical modelling.

Analog computers are convenient to run. Also, they present solutions in a simple and graphic manner in almost no time. Unfortunately, they are rather inaccurate and their versatility is limited. The class of analog computers includes electric integrators to solve partial differential equations, electronic machines to solve ordinary differential equations, machines to solve algebraic and transcendental equations, and special-purpose analog computers.

*Digital computers* deal with mathematical variables in the form of numbers which represent discrete values of physical quantities. Any number is represented as a combination of states of individual elements each of which can reside in several stable (or equilibrium) states. Digital computers are versatile, accurate and

can solve practically any mathematical problems and elaborate logical operations.

Although they operate at high rates, there is a limit to the speed of operation. This is because the time needed for a digital computer to solve a problem is usually a sum of that taken up by several non-computational operations, including control operations. The class of digital computers includes desk calculators, punch-card computers, and high-speed electronic digital computers.

If, in a system, analog and digital computers are combined, we talk of a *hybrid computer*. They combine the merits of both analog and digital machines.

In their evolution, digital and analog computers have been mechanical, then electromechanical, electrical and, finally, electronic. The advent of electronic computers in 1946-48 opened up wide vistas for fast and precise solution of mathematical and logical problems. In the Soviet Union, quite a number of electronic digital computers have appeared since then, including the Ural, the BESM, the Minsk, and the M-20. Major contributions to the advancement of digital computers have been made by a large team of Soviet scientists, among them Academician S. A. Lebedev, Yu. Ya. Basilevsky, B. I. Rameyev, and I. S. Brook.

Various physical principles have been used in the development of analog computers. In the second decade of the 20th century, analogs based on current-conducting paper and an electrolytic bath appeared. In the late 40's, a team under Professor L. I. Guttenmakher developed several electrical integrators to solve partial differential equations. Their invention was based on Professor S. A. Gershgorin's idea (1926) to use electric grids for modelling. Then came electronic mathematical models realized with d.c. amplifiers (ИПТ-4, ИПТ-5, МН-7, МН-14, МН-10, etc.). The theory and design of analog computers in the Soviet Union owe much to V. A. Trapeznikov, V. B. Ushakov, A. A. Feldbaum, N. Ye. Kobrinsky, A. N. Lebedev, and V. B. Smolov, to name but a few.

The 25th Congress of the CPSU has mapped out a far-reaching program of further development for national economy. It can successfully be achieved only through large-scale mechanization and automation of production, integration of the latest advances of science and technology in all fields of industry, and streamlined planning and management. This is an encouraging challenge to computers and computer engineering.

# Part One

## Analog Computers

### Chapter 1

#### Basic Theory of Similitude and Electric Modelling

##### 1.1. Basic Concepts of Similitude Theory

**1.1.1. Modelling methods.** Two broad methods of modelling are used most, physical and mathematical. In physical modelling, a model retains the physical character of the prototype; in mathematical modelling, the basis is the analogies that exist between the equations describing a prototype system and its model.

##### *Physical Modelling*

In physical modelling, one uses a model which is an exact or a scaled-down replica of the prototype and retains its physical character. The experimenter seeks to establish relationships between both the geometrical elements and the physical variables of the model and the prototype.

Let, for example, there be two electrical circuits which can be described by equations of the form

$$f(V, R, i, L) = 0, \quad f(V_1, R_1, i_1, L_1) = 0 \quad (1.1)$$

In the case of complete correspondence between the two circuits, their respective variables will be connected by the following relations

$$V/V_1 = M_V, \quad R/R_1 = M_R, \quad i/i_1 = M_i, \quad L/L_1 = M_L \quad (1.2)$$

where  $M_V$ ,  $M_R$ ,  $M_i$  and  $M_L$  are the similarity scale factors. Since in physical modelling analogous quantities have the same dimensions, the similarity scale factors are dimensionless.

Physical models give a full representation of the process or situation under study, because they can replicate both the mathematical and physical aspects of the prototype. However, they are used on a limited scale because of the difficulties in their construction, hard-and-fast specialization, and also because the quantities of interest are not at all easy to measure in some divisions of physics.

Physical modelling should preferably be used in situations where the process or system of interest does not lend itself to an

accurate mathematical description or where such a description leads to relationships of a complexity such that other methods may fail in mechanizing them. Examples of physical models are scale models of ships, aircraft, missiles, and water-power works.

### Mathematical Modelling

Mathematical models only represent the mathematical aspects of the process or system of interest. Although the equations applicable to both the prototype and the model must be identical in form, they may use entirely different physical quantities. Let, for example, the equation describing the behaviour of a mechanical system have the form

$$F = m \frac{du}{dt} + wu + \frac{1}{f} \int u dt \quad (1.3)$$

where  $F$  = force

$m$  = mass

$u$  = velocity

$w$  = friction coefficient

$f$  = elasticity

$t$  = time

and the behaviour of an electric model be described by an equation of the form

$$V = L \frac{di}{dt} + Ri + \frac{1}{C} \int i dt \quad (1.4)$$

where  $V$  = voltage

$L$  = inductance

$i$  = current

$R$  = resistance

$C$  = capacitance

$t$  = time

As is seen, both equations, (1.3) and (1.4), are the same in form. The relations that exist between the analogous variables of the prototype and the model may be written as

$$\begin{aligned} F/V &= M_F, \quad m/L = M_m, \quad u/i = M_u, \quad w/R = M_w, \\ f/C &= M_f, \quad t_{mech}/t_{el} = M_t \end{aligned} \quad (1.5)$$

where  $M_F$ ,  $M_m$ ,  $M_u$ ,  $M_w$ ,  $M_f$  and  $M_t$  are the similarity scale factors.

If two or more physical processes or systems have the same mathematical representation, we have what is known as *equation isomorphism*. Precisely this equation isomorphism is at the basis of mathematical modelling. Any physical process or system in such a group may serve as a model of any other. Electric models have come into prominence because they are simple and inexpensive to build, their circuit parameters can readily be adjusted at

will, measurements can be made and recorded with sufficient accuracy, and their maintenance is negligible. Unfortunately, the accuracy is rather limited.

Mathematical modelling is versatile and is widely used in practice. It may be divided into two types: direct simulation, or simulation on the basis of direct analogies and simulation on the basis of equations, or equation-solving.

1. *Simulation on the basis of direct analogies.* A direct analogy model is remarkable in that the equation describing what happens in the analog is the same in structure as that of the prototype system. In other words, a *direct analogy* is established between the respective variables in the prototype and analog equations. In building direct analogy models, the first step is to list all physical elements that make up the system to be simulated. Next, an

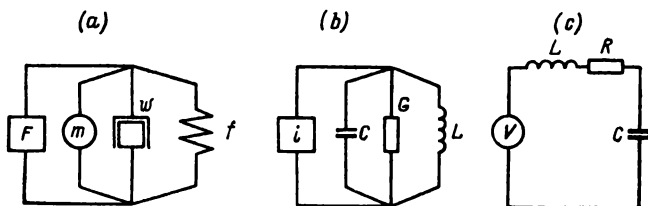


Fig. 1.1

analog is built for each element, and the individual analogs are interconnected so that the equation describing the behaviour of the overall model has the form of the prototype equation. With this approach, all physical elements are replicated by their respective analogs in a model. An added requirement is that each analog must be individually controllable so as to give freedom of variations during an experiment.

Consider an example of direct-analogy simulation. Let the prototype system be a one-degree-of-freedom mechanical system (Fig. 1.1a) described by Eq. (1.3). Electric analogs of the system may be the circuits (Fig. 1.1b and c) based on Kirchhoff's laws. The dynamic behaviour of the circuit shown in Fig. 1.1b (the nodal analogy) can be described as

$$i = C \frac{dV}{dt} + GV + \frac{1}{L} \int V dt \quad (1.6)$$

where  $i$ ,  $C$ ,  $G$  and  $L$  are the current source in and the capacitance, conductance and inductance of the circuit, each being analogous respectively to  $F$ ,  $m$ ,  $w$  and  $f$  of the prototype system (hence another name, the force-current analogy). The dynamic behaviour of the circuit shown in Fig. 1.1c (the loop analogy) can be described by Eq. (1.4), where  $V$ ,  $L$ ,  $R$  and  $C$  are the voltage

source in and the inductance, resistance and capacitance of the circuit, each being analogous respectively to the same four variables of the mechanical system (hence the name, the force-voltage analogy). The circuit parameters of the models are made variable so that they can be adjusted in the course of study. With the analogies thus derived, the prototype equation can be solved directly. In the loop analogy, the velocity  $u$  of the mechanical system is simulated by the voltage  $V$  across the electric circuit, and in the nodal analogy by the current  $i$  around the circuit.

Direct-analogy models are convenient where the experimenter is tracing the behaviour of a system in response to changes in its variables so as to determine their optimum values. Examples of direct-analogy models or simulators are resistance network analogs, electrolytic tanks and resistance-paper devices.

2. *Simulation on the basis of indirect analogies.* In contrast to direct-analogy simulation where physical elements of the prototype system are simulated, an indirect simulator only replicates the form of the prototype equation and the flow of computation. Indirect simulation uses analog computing elements (such as summers, integrators and inverters) which are interconnected to mechanize the problem equation. To optimize the problem set-up, the governing equation is often transformed to a point where a direct physical analogy between the variables of the prototype system and the coefficients of the transformed equation set up on the simulator is completely lost.

Consider an example of indirect simulation. We choose the same one-degree-of-freedom mechanical system (see Fig. 1.1a) to be the prototype system. Its behaviour may be described as

$$F = m \frac{d^2x}{dt^2} + w \frac{dx}{dt} + \frac{1}{f} x \quad (1.7)$$

This equation may be solved in several ways. One requires that Eq. (1.7) be re-written to solve for the highest derivative

$$\frac{d^2x}{dt^2} = -\frac{w}{m} \frac{dx}{dt} - \frac{1}{fm} x + \frac{1}{m} F \quad (1.8)$$

From Eq. (1.8) it follows that in order to obtain the sought variable  $x$ , we must add together the terms on the right-hand side and integrate the sum twice. The set-up required to mechanize this problem (Fig. 1.2) consists of two electric integrators,  $I_1$  and  $I_2$ , an inverter, and three potentiometers  $P_1$ ,  $P_2$  and  $P_3$ , and fits the equation

$$\frac{d^2V_x}{dt^2} = -a \frac{dV_x}{dt} - bV_x + cV_F \quad (1.9)$$



where  $V_x, V_F$  = machine analogs of  $x$  and  $F$ , respectively

$a, b, c$  = constant coefficients

By choosing appropriate scaling factors which relate each machine variable to a corresponding variable of the prototype system

$$M_x = x/V_x, M_F = F/V_F, M_t = t/t_m$$

we can readily pass from the machine solution

$$V_x = f(t_m)$$

to the sought variable,  $x = f(t)$ .

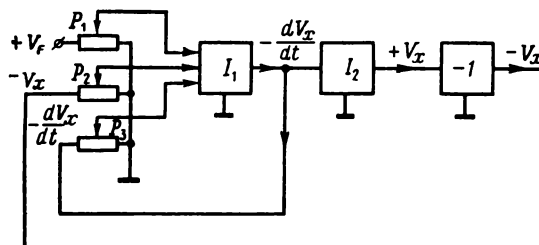


Fig. 1.2

**1.1.2. Methods for determining similarity variables.** Analogous systems must have identical similarity variables,  $\Pi$ , dimensionless groups of physical quantities, which are characteristic of a given process and which have the same value in the model and the prototype system ( $\Pi = \text{idem}$ ). Once similarity variables are known, one can readily pass to the relationships between the physical quantities of the prototype system and its model.

### Equation Analysis Method

This method is based on the proof of the first similarity theorem. Each equation under comparison is reduced to dimensionless form by dividing it through by some term. Then the signs of integration and differentiation are dropped in the transformed equation, and the terms of the equation thus derived give  $s - 1$  similarity variables, where  $s$  is the number of terms in the original equation.

Let the original equation be of the form

$$L \frac{di}{dt} + Ri + \frac{1}{C} \int i dt = V \quad (1.10)$$

and that of the model

$$C \frac{dV}{dt} + GV + \frac{1}{L} \int V dt = i \quad (1.11)$$

To reduce the two equations to dimensionless form, we divide each through by its right-hand side:

$$\frac{L}{V} \frac{di}{dt} + \frac{Ri}{V} + \frac{1}{CV} \int i dt = 1 \quad (1.12)$$

$$\frac{C}{i} \frac{dV}{dt} + \frac{GV}{i} + \frac{1}{Li} \int V dt = 1 \quad (1.13)$$

On dropping the signs of integration and differentiation, we get  $s - 1$  similarity variables for each equation. Since  $s = 4$ , then  $s - 1 = 3$ . For the systems to be analogous, the similarity variables must be identical:

$$\begin{aligned} \Pi_1 = Li/Vt = CV/it = \text{idem}, \quad \Pi_2 = Ri/V = GV/i = \text{idem} \\ \Pi_3 = it/CV = Vt/Li = \text{idem} \end{aligned} \quad (1.14)$$

Sometimes, the above method is referred to as the integral analog method. It applies to both homogeneous and inhomogeneous equations. In the latter case, the arguments of inhomogeneous functions in the prototype system and the model must be equal. For example, for the prototype and model equations of the form

$$q_i = a \tan \omega t$$

$$\Theta_i = k \tan \omega t$$

the similarity condition will be the equation

$$\omega t = \omega t$$

### Dimensional Analysis Method

This method is based on the second similarity theorem and consists in the following. Of the number  $m$  physical quantities characterizing a system, we arbitrarily choose  $k$  quantities having independent dimensions, such that  $k \leq q$  where  $q$  is the number of fundamental quantities used in the adopted system of units. For example, the SI system has four fundamental quantities, namely length,  $l$ , mass,  $M$ , time,  $t$ , and current,  $i$ ; that is,  $q = 4$ .

On selecting  $k$  out of  $m$  physical quantities, transforms of  $m - k$  similarity variables are obtained. Each similarity variable may be expressed as a fraction whose numerator is one of the  $(m - k)$  quantities raised to power  $+1$ , and whose denominator is the product of the  $k$  selected quantities by the yet unknown exponents,  $x_1, x_2, \dots, x_k, z_1, z_2, \dots, z_k$ , etc. The transforms of the similarity variables may be written as

$$\Pi_i = \frac{j_{k+1}}{j_1^{x_1} j_2^{x_2} \dots j_k^{x_k}}, \dots, \Pi_{m-k} = \frac{j_m}{j_1^{z_1} j_2^{z_2} \dots j_k^{z_k}} \quad (1.15)$$

The unknown exponents are found by solving the dimensional equation for each similarity variable. As an illustration, consider an example.

Let the system under study be described by an equation of the form

$$f(i, V, R, C, L, t) = 0$$

It is known that in this system  $k = 3$ . We choose the quantities  $V$ ,  $L$  and  $C$ . Then the number of similarity variables will be

$$m - k = 6 - 3 = 3$$

The transforms of the three similarity variables may be written as

$$\Pi_1 = \frac{i}{V^{x_1} L^{x_2} C^{x_3}}, \quad \Pi_2 = \frac{R}{V^{y_1} L^{y_2} C^{y_3}}, \quad \Pi_3 = \frac{t}{V^{z_1} L^{z_2} C^{z_3}} \quad (1.16)$$

The exponents  $x_i$ ,  $y_i$  and  $z_i$  may be evaluated on recalling that the numerator and denominator of a similarity variable have the same dimensions (this is why similarity variables are dimensionless). Therefore,

$$\begin{aligned} [i] &= [V]^{x_1} [L]^{x_2} [C]^{x_3} \\ [R] &= [V]^{y_1} [L]^{y_2} [C]^{y_3} \\ [t] &= [V]^{z_1} [L]^{z_2} [C]^{z_3} \end{aligned} \quad (1.17)$$

Equations (1.17) have been derived on the basis of the following point of dimensional analysis: the dimension of any derivative is the product of the dimensions of the fundamental quantities raised to certain powers, that is,

$$[r] = [a]^{a_1} [b]^{a_2} \dots [q]^{a_q}$$

where  $r$  is a derivative of a physical quantity, and  $a$ ,  $b$ , ...,  $q$  are the fundamental physical quantities in the system of units adopted (in our case, the SI system). To solve the first line in Eqs. (1.17), we write it in terms of the dimensions of the fundamental quantities adopted in the SI system:

$$[i^0 M^0 t^0 i^1] = [i^2 M^1 t^{-3} i^{-1}]^{x_1} [i^2 M^1 t^{-2} i^{-2}]^{x_2} [i^{-2} M^{-1} t^4 i^2]^{x_3} \quad (1.18)$$

where  $l$  = length in metres

$m$  = mass in kilograms

$t$  = time in seconds

$i$  = current in amperes

The exponents are taken from the respective dimensional tables of the SI system. The sum of exponents for each fundamental quantity in such an equation must be equal to zero. By writing

such equations of exponent balance for each fundamental quantity, we obtain the following system of equations

$$\begin{array}{r|l} 2x_1 + 2x_2 - 2x_3 = 0 & l \\ x_1 + x_2 - x_3 = 0 & M \\ -3x_1 - 2x_2 + 4x_3 = 0 & t \\ -x_1 - 2x_2 + 2x_3 = 1 & i \end{array} \quad (1.19)$$

whence

$$x_1 = -1, \quad x_2 = +\frac{3}{2}, \quad x_3 = +\frac{1}{2}$$

Similarly, we can evaluate the exponents of the remaining similarity variables:

$$\begin{array}{ll} y_1 = 0 & z_1 = 0 \\ y_2 = 1/2 & z_2 = 1/2 \\ y_3 = -1/2 & z_3 = 1/2 \end{array}$$

Then, according to Eqs. (1.16), the similarity variables may be written as

$$\Pi_1 = \frac{iV}{\sqrt{L^3C}} = \text{idem}, \quad \Pi_2 = \frac{R\sqrt{C}}{\sqrt{L}} = \text{idem}, \quad \Pi_3 = \frac{t}{\sqrt{LC}} = \text{idem} \quad (1.20)$$

Further similarity variables may be obtained by multiplying and dividing the variables  $\Pi_1$  through  $\Pi_3$ , Eqs. (1.20), by one another. The variables thus obtained can often give a deeper insight into the physical significance of similarity.

## 1.2. Synthesis of Electric Models for Physical Systems

**1.2.1. Synthesis of physical models.** The development of an electric model involves two basic steps. The first defines its circuit configuration, and the second yields its circuit parameters.

Consider an example. Let the prototype system be the circuit of Fig. 1.3. The numerical values of the circuit parameters are:

$$V_1 = 1000 \sin 314t \text{ V}, \quad V_2 = 300 \sin 1570t \text{ V}$$

$$L_1 = 0.1\text{H}, \quad L_2 = 0.2\text{H}$$

$$C_1 = 100\mu\text{F}, \quad C_2 = 200\mu\text{F}$$

$$R_1 = 5 \text{ ohms}, \quad R_2 = 10 \text{ ohms}, \quad R_3 = 1000 \text{ ohms}$$

Construct the analog circuit. In physical modelling, the analogous quantities in the prototype and the model are the same physically and dimensionally. Therefore, the analog circuit will be an exact copy of the prototype circuit and will only differ in the numerical

values of the circuit parameters (in Fig. 1.3, the model parameters are enclosed in parentheses).

The circuit parameters for the analog circuit are evaluated as follows. One of the parameters of each dimension is selected as the basis quantity for the remaining parameters of the same dimension. In our example, we choose the basis quantities to be:

$$V_0 = 1000 \text{ V}, \quad L_0 = 0.1 \text{ H}$$

$$i_0 = 1 \text{ A}, \quad \omega_0 = 314 \text{ rad/s}$$

$$C_0 = 100 \mu\text{F}, \quad t_0 = 1 \text{ s}, \quad R_0 = 10 \text{ ohms}$$

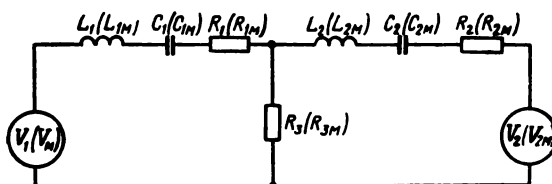


Fig. 1.3

Dividing all the parameters of the prototype circuit by the selected basis quantities yields a dimensionless (scaled) representation of the prototype circuit:

$$\begin{aligned} M_{V_1} &= V_1/V_0, & M_{V_1} &= 1, & M_{V_2} &= 0.3 \\ M_{\omega} &= \omega_1/\omega_0 = 1, & M_{\omega_2} &= 5 \\ M_{L_1} &= L_1/L_0 = 1, & M_{L_2} &= 2 \\ M_{C_1} &= C_1/C_0 = 1, & M_{C_2} &= 2 \\ M_{R_1} &= 0.5, & M_{R_2} &= 1, & M_{R_3} &= R_3/R_0 = 100 \end{aligned} \quad (1.21)$$

Then we choose an arbitrary number  $k$  of independent basis quantities for the model; in our case,  $k = 3$ . Let them be  $V_{0m} = 10 \text{ V}$ ,  $C_{0m} = 1 \mu\text{F}$ , and  $L_{0m} = 0.1 \text{ H}$ . In order to evaluate the remaining basis quantities for the model, we should first find similarity variables. For the system in question, one of the alternative forms is

$$\Pi_1 = Vt/Li, \quad \Pi_2 = V/Ri, \quad \Pi_3 = VC/it, \quad \Pi_4 = \omega t \quad (1.22)$$

Substituting the known basis quantities of the prototype circuit and the arbitrarily chosen basis quantities for the model in

Eq. (1.22) gives

$$\begin{aligned}\Pi_1 &= \frac{1000 \times 1}{0.1 \times 1} = \frac{10t_{0m}}{0.1i_{0m}} \\ \Pi_2 &= \frac{1000}{10 \times 1} = \frac{10}{R_{0m}i_{0m}} \\ \Pi_3 &= \frac{1000 \times 100 \times 10^{-6}}{1 \times 1} = \frac{10 \times 1 \times 10^{-6}}{i_{0m}t_{0m}} \\ \Pi_4 &= 314 \times 1 = \omega_{0m}t_{0m}\end{aligned}$$

The remaining basis quantities are found to be:

$$t_{0m} = 0.1 \text{ s}, \quad i_{0m} = 1 \text{ mA}, \quad R_{0m} = 100 \text{ ohms}, \quad \omega_{0m} = 3140 \text{ rad/s}$$

After all basis quantities of the model have been found, its circuit parameters are evaluated by multiplying the scale factor of each parameter in the prototype circuit by the respective basis quantity in the model. As a result, we get

$$L_{1m} = M_{L1}L_{0m} = 0.1 \text{ H}$$

$$L_{2m} = 0.2 \text{ H}$$

$$R_{1m} = 50 \text{ ohms}$$

$$R_{2m} = 100 \text{ ohms}, \quad R_{3m} = 10^4 \text{ ohms}$$

$$C_{1m} = 1 \mu\text{F}, \quad C_{2m} = 2 \mu\text{F}$$

$$V_{1m} = 10 \text{ V}, \quad V_{2m} = 3 \text{ V}$$

$$\omega_{1m} = 3140 \text{ rad/s}, \quad \omega_{2m} = 15,700 \text{ rad/s}$$

**1.2.2. Synthesis of mathematical models.** In mathematical modelling, the equations of the prototype system and the model are the same in form, but the analogous quantities may differ in nature and dimensions. The examples that follow will throw enough light on how electric direct analogs are synthesized.

**Example 1.** Let the prototype system be the circuit shown in Fig. 1.1c. The numerical values of the circuit parameters are:

$$V = 1000 \sin 314t \text{ V}, \quad L = 1 \text{ H}, \quad R = 10 \text{ ohms}, \quad C = 250 \mu\text{F}$$

Before going any further, it will be instructive to learn some basic facts about dual networks. Suppose we have two networks such that the mesh currents in one obey the Kirchhoff current law,  $\Sigma i = 0$ , which is the same in form as the Kirchhoff voltage law governing the node voltages,  $\Sigma V = 0$ , in the other. Then one circuit of the pair is said to be the *dual* of the other. As already noted, the equations expressing the two laws are identical in form, but the analogous terms have different dimensions.

Going back to the circuit of Fig. 1.1*b*, its equation of current balance is

$$i = C \frac{dV}{dt} + GV + \frac{1}{L} \int V dt \quad (1.23)$$

while the voltage balance equation for the circuit of Fig. 1.1*c* is

$$V = L \frac{di}{dt} + Ri + \frac{1}{C} \int i dt \quad (1.24)$$

that is, the two equations are identical in form, which is another way of saying that the circuits of Fig. 1.1*b* and *c* are *duals*.

Now, if one of the duals is chosen to be the prototype, the other (dual) network will be its mathematical model. The model parameters can be found in the same manner as for a physical model. Reducing the prototype network to dimensionless form yields a scaling factor of unity for each parameter, that is

$$M_V = M_L = M_C = M_R = M_t = 1$$

Next, we choose an arbitrary number  $k$  of basis quantities for the model. In our example,  $k = 3$ , and the basis quantities may well be  $i_{0m}$ ,  $C_{0m}$ , and  $L_{0m}$ . Let  $i_{0m} = 10$  mA,  $C_{0m} = 1$   $\mu$ F, and  $L_{0m} = 0.1$  H. In one of the alternative forms, the similarity variables may then be written as

$$\begin{aligned} \Pi_1 &= Li/Vt = CV/it, & \Pi_2 &= Ri/V = GV/i \\ \Pi_3 &= it/CV = Vt/Li, & \Pi_4 &= \omega t = \omega_m t_m \end{aligned} \quad (1.25)$$

Substituting the basis quantities of the prototype network and the arbitrarily selected basis quantities for the model in Eqs. (1.25) leads to a system of equations whose solution yields the remaining basis quantities for the model:

$$\begin{aligned} V_{0m} &= 5 \text{ V} \\ t_{0m} &= 0.5 \text{ s} \\ G_{0m} &= 2 \times 10^{-4} \text{ mA} \\ \omega_{0m} &= 628 \text{ rad/s} \end{aligned}$$

whence

$$\begin{aligned} i_m &= M_V i_{0m} = 10 \text{ mA} \\ C_m &= M_C C_{0m} = 1 \mu\text{F} \\ L_m &= M_L L_{0m} = 0.1 \text{ H} \\ G_m &= M_R G_{0m} = 2 \times 10^{-4} \text{ ohm}^{-1} \end{aligned}$$

**Example 2.** Let the prototype system be a one-degree-of-freedom mechanical system, described by an equation of the form

$$F = m \frac{du}{dt} + \omega u + \frac{1}{f} \int u dt \quad (1.26)$$

An electric model for the prototype mechanical system can be constructed on the basis of well-known electric-mechanical analogies. Thus, the force balance in mechanics corresponds to voltage and current balances in electric systems. Hence, two sets of analogous quantities (loop and nodal analogies) in mechanical and electric systems may be constructed as shown in Table 1.1.

Table 1.1

Mechanical system	Electric analogies	
	loop analogies	nodal analogies
$F$	$V$	$i$
$m$	$L$	$C$
$w$	$R$	$G$
$f$	$C$	$L$
$u$	$i$	$V$

Accordingly, two circuit configurations may be obtained for the same prototype system. The actual procedure is as follows. In Eq. (1.26) of the prototype system, all mechanical elements are replaced with their electric analogs from one of the two sets of analogies. This gives two analogous equations for the system:

$$i = C \frac{dV}{dt} + GV + \frac{1}{L} \int V dt \quad (1.27)$$

$$V = L \frac{di}{dt} + Ri + \frac{1}{C} \int i dt \quad (1.28)$$

The circuits constructed to satisfy Eqs. (1.27) and (1.28) will have the form shown in Fig. 1.1*b* and *c*. These circuits are duals because they are described by two analogous equations, one representing an application of Kirchhoff's voltage law and the other that of Kirchhoff's current law.

In mechanics, as in electrical engineering, the dynamic behaviour of a system may be brought out through the use of some idealised elements. For example, the dynamic behaviour of a mechanical system may be depicted as shown in Fig. 1.1*a*. From a comparison of this diagram with those shown in Fig. 1.1*b* and *c*, it can readily be recognized that it is analogous to the circuit of Fig. 1.1*b*. This correspondence suggests one more method for building an electric model: replacing all elements in the dynamic representation of the prototype mechanical system with their electric analogies from the second set will lead to one more electric model. Its parameters can be found by the methods already explained.



In simulation on the basis of indirect analogies (by equation solving) the characteristic equation of the prototype system is arrived at by consecutively performing all the necessary mathematical steps. Such models are synthesized each time a problem is to be solved, by arranging the computing elements required to solve the equation into a block diagram. Then scale factors relating each variable in the prototype system to a corresponding variable within the computer are determined, the unit gains are adjusted, and the initial conditions and external disturbances are set in. Now the problem is ready to be set up on an analog computer. In greater detail, the construction of block diagrams on the basis of indirect simulation is explained in Chap. III.

### 1.3. Electric Modelling of Physical Fields

**1.3.1. Basic concepts.** Physical-field problems may be classed into internal and external. *Internal-field* problems arise in cases where the field boundaries are rigorously defined. In solving an internal field problem, one is interested in the behaviour of a sought variable within a region under given boundary conditions and a specified process in that region.

Solving an internal field problem involves the knowledge of: (a) the region where the process of interest takes place (usually in the form of a drawing); (b) the nature of the process (defined by an equation); (c) initial conditions of the field (to be specified only for nonstationary or time-varying fields); (d) boundary conditions as appropriate potentials at the field boundary.

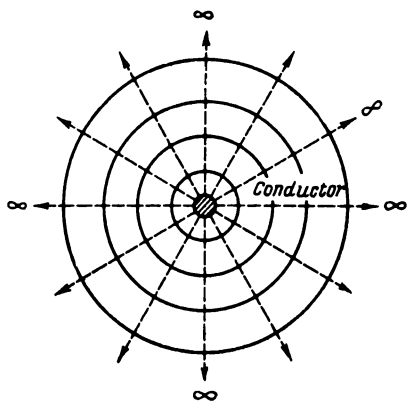


Fig. 1.4

Boundary conditions may be of the first, second or third kind. Boundary conditions of the first kind are specified as values of functions  $\phi$  at points on the boundary. This constitutes the first boundary-value, or Dirichlet, problem. Boundary conditions of the second kind are given by specifying the normal derivative of the sought variable,  $d\phi/dn$ , on the boundary. This is the second boundary-value, or Neumann, problem. In such a case, the value of the sought function must be known at one point within the region or on the boundary as a minimum. Boundary conditions of the third kind are specified by a linear relation between the

sought function and its normal derivative

$$C = a\varphi + b \frac{d\varphi}{dn}$$

Solving boundary-value problems gives the function distribution within the region.

*External-field* problems involve a field external with respect to some object, for example, the magnetic field around a conductor (Fig. 1.4). An external field is usually infinite. In such fields, a source of excitation is an object placed somewhere within (usually, at the centre of) the field. To solve an external-field problem, the characteristic equation of the process must be specified. Its solution yields the distribution of the sought function in the external field.

Field problems may be stationary, or time-invariant, and non-stationary, or time-varying. In the former case, one deals with steady-state processes, in the latter with nonsteady-state. Field problems are formulated as partial differential equations. Those applying to stationary cases are Laplace's and Poisson's equations (in three-dimensional space):

$$\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} + \frac{\partial^2 \varphi}{\partial z^2} = 0$$

$$\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} + \frac{\partial^2 \varphi}{\partial z^2} = f(x, y, z)$$

also called elliptic partial differential equations. Those applying to nonstationary cases are called parabolic partial differential equations. They are exemplified by Fourier's heat-flow equation

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} = f(x, y, z) \frac{\partial T}{\partial t}$$

by the wave equation (describing wave motion in various media)

$$\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} + \frac{\partial^2 \varphi}{\partial z^2} = k \frac{\partial^2 \varphi}{\partial t^2}$$

by the biharmonic equation (elastic vibrations of a membrane)

$$\frac{\partial^4 \varphi}{\partial x^4} + 2 \frac{\partial^4 \varphi}{\partial x^2 \partial y^2} + \frac{\partial^4 \varphi}{\partial y^4} = 0$$

Field problems may be solved by general analog techniques, both physical and mathematical. Most often, resort is made to electric-field analogs.

**1.3.2. Continuous field analogs.** This type of analog uses a field due to a current flowing through a continuous conducting material, such as resistance paper, metal foil, electrolyte in a tank, conducting colloids, and the like. The material must have as high a resistivity as practicable. As often as not, the material is a

semiconductor. The greater the resistivity of the material, the greater the absolute value of field gradients and the more accurate the results.

With a conductive-sheet analog, the sheet has the same geometrical shape (or a conformal transformation thereof) as the field under study. The boundary conditions of the original field are simulated in the analog system by application of voltage sources (in the case of boundary conditions of the first kind) or current sources (in the case of boundary conditions of the second kind) to electrodes in contact with the sheet at the boundaries. The voltage distribution on the surface of the conductive sheet

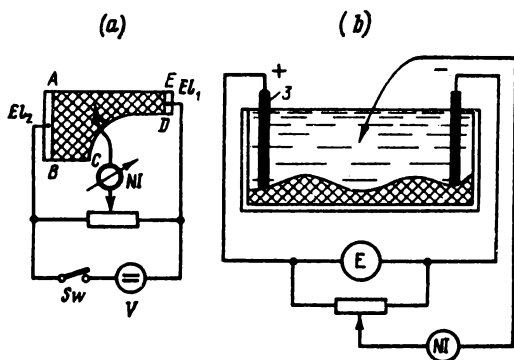


Fig. 1.5

is then measured and recorded by means of suitable sensing equipment (usually a potentiometric field plotter).

Examples of continuous field analogs appear in Fig. 1.5. That in Fig. 1.5a is a resistance-paper analog of a uniform two-dimensional physical field, governed by Laplace's equation. Boundary conditions of the first kind are simulated at boundaries AB and DE by application of voltages from a source to electrodes  $EL_1$  and  $EL_2$ . The remaining part of the boundary is not connected to a boundary-condition source, which implies that zero values of the normal derivatives are specified.

Analogues for nonuniform fields may be constructed by using pieces of different resistance paper glued together, or by painting or touching up the sheet. Electrodes made from copper sheet 0.1 to 0.2 mm thick are usually bonded to the paper sheet with a current-conducting cement.

Figure 1.5b shows a conducting-liquid analog of a uniform three-dimensional field governed by Laplace's equation. Generally called an electrolytic tank, it has one of its surfaces shaped to be a scale model of the boundary configuration of the field

under study. The material for the boundary model is usually a mixture of paraffin and rosin or wax.

For their operation, electrolytic tanks depend on the ion conduction displayed by solutions of common salt, sulphuric acid or copper sulphate in water. The conduction of the electrolyte may be varied within wide limits by suitably changing the salt or acid concentration. The tank itself may be deep or shallow. In the latter case, the electrolyte depth is held constant at a few centimetres.

The voltage distribution in the liquid is sensed by a probe or an array of probes connected to suitable measuring instruments. Each probe is mounted in a carriage which is installed over the tank and can be positioned as required. The position of the probe within the tank is transferred by a pantograph onto a drawing which represents the surface of the electrolyte on a full or a reduced scale. By guiding the probe in such a fashion that the galvanometer (null indicator) reads zero at all times for a given potentiometer setting, one obtains so-called equipotential lines. From a family of equipotential lines, one can readily derive a set of stream lines representing the flow of current through a region of the field and normal to the equipotential lines. The complete solution of a field problem is then a grid of orthogonal equipotential and stream lines. The plotting of this grid (field plotting) may be automated.

The use of conducting liquids for three-dimensional field analogs, such as in electrolytic tanks, is explained by the need to have access to any point within a region to measure the potential with a probe. Instead of an electrolyte, the conducting material may be a colloidal mass.

Field simulation advantageously utilizes the property of field symmetry. If the field is symmetrical, only part of it, bounded by the symmetry axes, need be simulated.

Among the advantages of continuous-type field analogs are simplicity, a relatively low cost, and the possibility of measuring the sought function at any point within the simulated field. On the negative side are variations in the properties of the conducting material, dependence on ambient conditions, and a limited range of applications. In fact, they are only suitable to solve problems governed by Laplace's equation and, though to a lower degree of accuracy, by Poisson's equation. An example of Soviet-made commercial continuous-type field analogs is the *EGDA* series using several grades of resistance paper.

**1.3.3. Resistance-network method.** This method is based on the formal similarity between Kirchhoff's law equations for resistance networks and the equations resulting from the approximation of

the partial differential equations describing field problems with a set of finite-difference equations.

Suppose we want to solve a problem involving a stationary uniform field governed by Laplace's equation in two dimensions

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \quad (1.29)$$

where  $T$  = temperature

$x$  and  $y$  = coordinates of a thermal field

To solve the problem by the resistance network method, Eq. (1.29) is approximated by finite differences,

$$\frac{\Delta^2 T}{\Delta x^2} + \frac{\Delta^2 T}{\Delta y^2} = 0 \quad (1.30)$$

The first difference,  $\Delta T$ , is the difference in value of the function (the temperature  $T$ ) between two adjacent points  $a$  and  $b$  of the

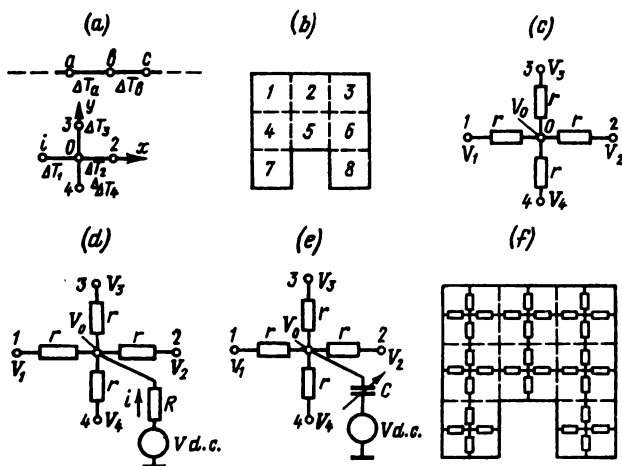


Fig. 1.6

field (Fig. 1.6a):

$$\Delta T_a = T_a - T_b$$

The second difference is the difference of the first differences between two adjacent line segments,  $ab$  and  $bc$ :

$$\Delta^2 T = \Delta T_a - \Delta T_b = (T_a - T_b) - (T_b - T_c) = T_a + T_c - 2T_b$$

For

$$\Delta x = \Delta y = h, \quad \Delta T_1 = T_1 - T_0, \quad \Delta T_2 = T_0 - T_2$$

$$\Delta T_3 = T_3 - T_0, \quad \Delta T_4 = T_0 - T_4$$

we have

$$\Delta^2 T_x = \Delta T_1 - \Delta T_2 = (T_1 - T_0) - (T_0 - T_2)$$

$$\Delta^2 T_y = \Delta T_3 - \Delta T_4 = (T_3 - T_0) - (T_0 - T_4)$$

Then Eq. (1.30) takes the form

$$T_1 + T_2 + T_3 + T_4 = 4T_0 \quad (1.31)$$

that is, the differential equation, Eq. (1.29), describing the field may be approximated by an algebraic equation, Eq. (1.31).

The resistance-network method of solution proceeds as follows. The continuous field to be simulated is discretized, that is, replaced by an array of discretely spaced elements (elementary parallelepipeds for a three-dimensional field, areas for a two-dimensional field, and segments for a one-dimensional field), referred to as a finite-difference grid. Suppose the original continuous field is in two dimensions and has the shape shown in Fig. 1.6*b*. We divide it into eight elementary areas, 1 through 8. For each area, an equivalent network of resistors is then constructed, with a dynamic behaviour described by the finite-difference equation of the original field. For example, if the dynamic behaviour of the original field is governed by Laplace's equation, Eq. (1.29), its finite-difference approximation will have the form of Eq. (1.31), and each node of the two-dimensional field will be simulated by the resistance-network junction shown in Fig. 1.6*c*. This network junction is described by

$$V_1 + V_2 + V_3 + V_4 = 4V_0 \quad (1.32)$$

If the dynamic behaviour of the field is governed by Poisson's equation

$$\frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} = F(x, y)$$

its finite-difference approximation will be

$$\Phi_1 + \Phi_2 + \Phi_3 + \Phi_4 = 4\Phi_0 + h^2 F(x, y) \quad (1.33)$$

and the respective resistance-network junction answering Eq. (1.33) will be as shown in Fig. 1.6*d*, and its behaviour will be described by

$$[V_1 + V_2 + V_3 + V_4]q = 4V_0q + i(x, y)h^2$$

for  $R \gg r$  and  $q = 1/r$ .

If the process within a field is governed by Fourier's equation

$$\frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} = F(x, y, t)$$

its model will be governed by an equation of the form

$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = R_0 C_0 \frac{\partial V}{\partial t}$$

As a rule, in resistance-network simulation the left-hand side of Eq. (1.32) is presented in finite-difference form and the right-hand side in differential form (as a capacitance-charging circuit). The typical resistance-network junction for such a field is shown in Fig. 1.6e.

The resistance-network junctions derived as explained above, are interconnected in the same manner as the nodal points of the original field, to form a resistance-network analog. Such an analog of a Laplacian field appears in Fig. 1.6f. Then, the resistance network is excited at its boundaries to simulate boundary

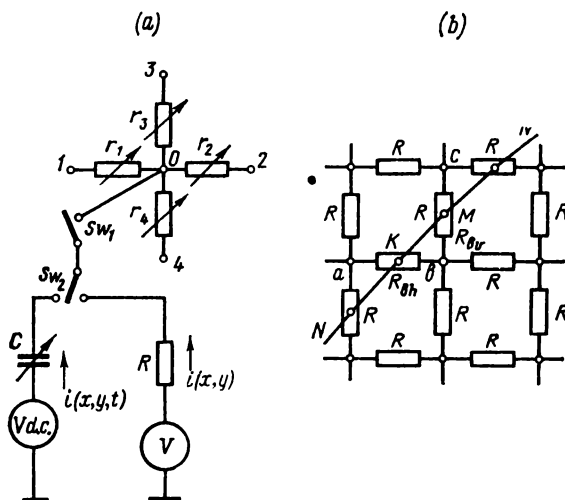


Fig. 1.7

conditions with suitable voltage sources in the case of a Dirichlet problem or with suitable current sources in the case of a Neumann problem. If the field being solved is nonstationary or, which is the same, the sought variable is varying with time and position, initial conditions must be simulated in the resistance network prior to solving the problem, with suitable voltage sources connected to all junction points of the analog, according to the initial values of the function. The solution is obtained by measuring and recording the voltages appearing at the junctions of the network. These voltages are then proportional to the potentials at the corresponding nodal points in the original field.

Resistance-network simulation is more versatile than that with continuous-type analogs. Apart from Laplacian fields, it is applicable also to problems described by Poisson's, Fourier's, wave and other equations. The parameters of resistance-network models

can readily be varied because all analog elements are made adjustable.

In the Soviet Union, several types of resistance-network analogs, called electric-network integrators, are available commercially (ЭИ-12, ЭИ-22, УСМ, etc.). They may be set up to solve either a specific field problem or several types of problems. An elementary resistance network of a general-purpose integrator for two-dimensional fields has the form shown in Fig. 1.7a. The circuit can readily be modified with switches  $Sw1$  and  $Sw2$ , to obtain analogs for fields governed by Laplace's, Fourier's and other equations. The resistors and capacitors that make up the analog can be adjusted in value to reflect discontinuities

in the fields under study, and to approximate field boundaries because the conductance between two neighbouring junction points simulates the distance between the analogous nodal points of the original field. This distance (known as the grid size) can then conveniently be varied (as shown in Fig. 1.7b) by varying the conductances. The resistances on the boundary  $N$  of the region being simulated are found by interpolation from the relation

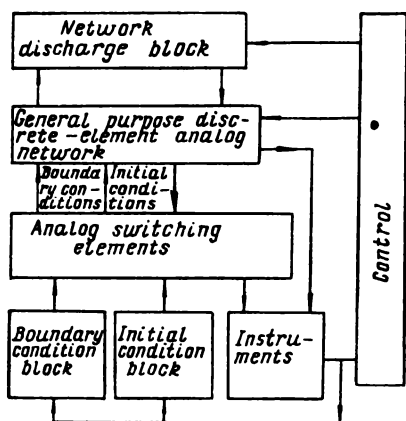


Fig. 1.8

$$R_{b.v} = R \frac{Mb}{cb}, \quad R_{b.h} = R \frac{kb}{ab}$$

where  $R$  = resistance of network elements within the region

$R_{b.v}$  = resistance on the vertical boundary

$R_{b.h}$  = resistance on the horizontal boundary

**Generalized block-diagram of an electric-network integrator.** Referring to Fig. 1.8, which is a generalized block-diagram, an electric-network integrator is made up of:

(1) *An electric network*, that is, an interconnection of a certain number of general-purpose discrete-type analog elements.

(2) *A boundary-condition block*, which is an assemblage of voltage and current sources, both fixed and functional. Boundary conditions of the first kind are simulated by directly applying appropriate voltages to the respective junction points on the boundaries from a voltage divider. Boundary conditions of the second kind are simulated by applying voltages to points on the boundaries via capacitors offering a higher impedance than the rest of the circuit. Boundary conditions of the third kind are





under boundary conditions of the first, second and third kind, where  $A_1(x, y)$  and  $A_2(x, y)$  are coefficients which are functions of the  $x, y$  coordinates, and also by Poisson's nonhomogeneous differential equations of the form

$$\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} = f(x, y) \quad (1.35)$$

or

$$\frac{\partial}{\partial x} \left[ A_1(x, y) \frac{\partial \varphi}{\partial x} \right] + \frac{\partial}{\partial y} \left[ A_2(x, y) \frac{\partial \varphi}{\partial y} \right] = f(x, y)$$

with the function  $\varphi$  set to zero on the region boundary and under boundary conditions of the second kind.

The resistor network, *RN*, consists of 940 variable resistors (in the form of resistance boxes) adjustable from 0 to 1000 ohms within the network region and from 0 to 10,000 ohms on the boundary. Boundary conditions are simulated by a circuit, *BCC*, incorporating an autotransformer divider, and initial conditions in the solution of Poisson's equations by a circuit, *ICC*, made up of an inductive source divider, *SD*, and a set of fixed capacitances, *SFC*, each equal to 0.02  $\mu\text{F}$  and connected to each junction point in the network.

The electric-network integrator operates on a.c. supply at commercial frequency. The elements are interconnected in any desired way by flexible cords on a patch board. The potentials at the junction points are measured or recorded with a meter *M*, or a recorder. The accuracy of the integrator is 0.1 to 2%.

Nonstationary fields described by, say, Fourier's equation of the form

$$\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} = f(x, y) \frac{\partial \varphi}{\partial t}$$

under specified boundary and initial conditions may be simulated on a Type ЭИ-22 electric-network integrator operating on direct current. The integrator has an electric network consisting of 450 variable resistors, a variable-capacitance box (adjustable from 0 to 5.55  $\mu\text{F}$ ), an initial- and boundary-condition unit, a display unit (oscilloscope), and an iterator. The latter enables the solution procedure to be iterated repeatedly for recording, because the transients are of short duration. The accuracy ranges from 1 to 5%.

The capabilities of the ЭИ-12 and the ЭИ-22 are combined in the ЭИ-31. The ЭП-41 can simulate three-dimensional Laplacian fields under boundary conditions of the second kind. The УСМ-1 simulates field problems described by elliptic and parabolic differential equations, and also by fourth-order equations in a three-coordinate system. The ЭИ-С is a special-purpose machine to si-

mulate processes in petroleum-bearing strata; its network has 13,000 junction points.

In recent years, much headway has been made in the development of mixed computing systems, such as the SATURN which incorporates an automatic electric-network analog machine and a general-purpose digital computer linked together by an interface. In addition to linear differential equations, such computing systems are capable of solving nonlinear partial differential equations in which the coefficients of the derivatives are dependent on position, time and the function itself (nonstationary problems in a three-dimensional region), that is

$$\begin{aligned} \frac{\partial}{\partial x} \left[ A_1(x, y, z, \varphi, t) \frac{\partial \varphi}{\partial x} \right] + \frac{\partial}{\partial y} \left[ A_2(x, y, z, \varphi, t) \frac{\partial \varphi}{\partial y} \right] \\ + \frac{\partial}{\partial z} \left[ A_3(x, y, z, \varphi, t) \frac{\partial \varphi}{\partial z} \right] = a^2(x, y, z, \varphi, t) \frac{\partial \varphi}{\partial t} \end{aligned}$$

under various boundary and initial conditions. In solving these problems, use is made of special methods for time discretization (for example, Liebmann's method), owing to which a nonlinear problem can be reduced to a linear one at each iteration. Resistances for the network are set, boundary and initial conditions are applied, results are measured and recorded, data are exchanged and all other operations are carried out automatically.

# Chapter II

## Computing Elements

### 2.1. General

**2.1.1. Scale factors.** In analog computation, the magnitude of each dependent variable of a problem is represented by some physical quantity, sometimes called the machine variable (a voltage  $V$ , a current  $i$ , or a displacement  $l$ ), in a computing element. The constant relating the number of units of the simulated quantity, called a *problem variable*, to one unit of the machine variable is called the *amplitude scale factor*,  $M$ , written as

$$M_x = \frac{x}{V_{in}} \frac{\text{units}}{\text{volt}}, \quad M_y = \frac{y}{V_{out}} \frac{\text{units}}{\text{volt}}, \quad M_z = \frac{z}{l} \frac{\text{units}}{\text{mm}} \quad (2.1)$$

To improve the accuracy of simulation, it is sought in choosing scale factors to utilize the entire range of values that the machine variables can take, that is,

$$M_x \geq \frac{|x|_{\max}}{|V_{in}|_{\max}}, \quad M_y \geq \frac{|y|_{\max}}{|V_{out}|_{\max}}, \quad \dots \quad (2.2)$$

**2.1.2. Scaling equations.** The relations between the input and output scale factors are expressed by *scaling equations*. Taking the simulation of a function  $z = xy$  by an electric multiplier in which

$$V_{out} = kV_1V_2 \quad (2.3)$$

where  $V_1 \propto x$ ,  $V_2 \propto y$ , and  $V_{out} \propto z$  are the input and output voltages, the scaling equation will have the form

$$M_z = \frac{1}{k} M_x M_y \quad (2.4)$$

It is obtained by substituting in Eq. (2.3) the voltages expressed in terms of the variables and scale factors according to Eq. (2.1) and by comparing the resultant expression with the specified function. Any two scale factors may be selected with relative freedom to satisfy the condition, Eq. (2.2), the third will then be decided by Eq. (2.4).

In analog computation, the original mathematical equations are modified into machine equations by means of scale factors (see Chap. III). Very often, the independent variable in these problems is time,  $t$ . Computer time,  $\tau$ , is related to problem time by the equation

$$t = M_t \tau$$

where  $M_t$  is the time scale factor. For  $M_t = 1$ ,  $t = \tau$ , and the computer is said to be operating on a real-time scale, that is, at the same rate as the process being simulated. A value of  $M_t$  greater than unity indicates that the computer operates faster than the actual system. A value of  $M_t$  smaller than unity indicates that the computer operates more slowly than the actual system.

## 2.2. Basic Elements of Computing Circuits

**2.2.1. Potentiometers.** These devices may be classed in several ways. According to the relation between the displacement of the wiper (or slider) and the output voltage, potentiometers are divided into linear and nonlinear. According to construction, potentiometers may be continuous-wound and sectionalized. Sectionalized potentiometers have not found any appreciable use in computing circuits because of limited accuracy and relatively complex design.

Consider a continuously-wound linear potentiometer. In the absence of loading (Fig. 2.1a), the output voltage is defined as

$$V_{out}^0 = \frac{r_x}{R} V_0 \quad (2.5)$$

where  $r_x$  = potentiometer setting (or dial reading) proportional to the independent variable

$R$  = total resistance of the potentiometer

$V_0$  = excitation voltage

Since

$$r_x = k_1 x$$

then

$$V_{out}^0 = \frac{k_1 V_0}{R} x = Cx$$

or, in words, the output voltage is a linear function of  $x$ .

If it is loaded (which is always the case in practical circuits), the output voltage of the potentiometer is given by

$$V_{out, loaded} = i R_{eq} = \frac{V_0}{R_t} R_{eq} \quad (2.6)$$

where  $i$  = current around the circuit

$R_{eq}$  = equivalent resistance equal to the potentiometer sett-

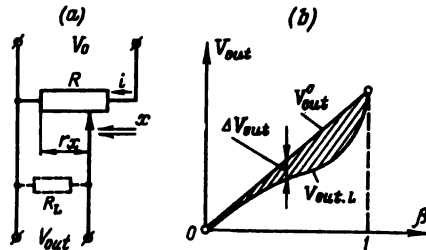


Fig. 2.1

ing,  $r_x$ , and the load resistance,  $R_L$ , connected in parallel

$R_t$  = total resistance

Substituting  $R_{eq} = r_x R_L / (r_x + R_L)$  and  $R_t = R - r_x + R_{eq}$  in the above expression gives

$$V_{out, loaded} = \frac{r_x R_L}{r_x (R - r_x) + R R_L} V_0 \quad (2.7)$$

on setting

$$r_x / R = \beta \quad \text{and} \quad R / R_L = \alpha$$

and dividing by  $R R_L$ , we get

$$V_{out, loaded} = \frac{\beta}{\alpha \beta (1 - \beta) + 1} V_0 \quad (2.8)$$

The response curves of a potentiometer, in the absence of loading and with the load resistance connected across the output terminals, are shown in Fig. 2.1*b*. The error in output voltage due to loading (the loading correction) is defined as

$$\delta V_{out} = \frac{V_{out, loaded} - V_{out}^0}{V_0} = - \frac{\alpha \beta^2 (1 - \beta)}{\alpha \beta (1 - \beta) + 1} \quad (2.9)$$

In practical circuits, it is usual to have  $\alpha \leq 0.1$  to  $0.01$  for  $\delta V_{out} \leq 0.02$ , and  $\beta < 1$ ; therefore,

$$\alpha \beta (1 - \beta) \ll 1$$

and Eq. (2.9) may be simplified as

$$\delta V_{out} \approx - \alpha \beta^2 (1 - \beta) \quad (2.10)$$

On differentiating this expression with respect to  $\beta$  and equating the derivative to zero, we find that the error will be a maximum at  $\beta = 0.7$ . Substituting this value in Eq. (2.10) yields

$$(\delta V_{out})_{max} \approx 0.15 R / R_L \quad (2.11)$$

Using Eq. (2.11), one can select parameters for a loaded linear potentiometer from the specified loading correction or find the loading correction from the values of  $R$  and  $R_L$  selected by other considerations.

Several methods exist to reduce the effect of loading (for example, insertion of a buffer amplifier between the potentiometer and its load, provision of a shaped-card or tapered compensating potentiometer in series with  $R$ , etc.).

**2.2.2. Synchro resolvers.** *Synchro (or induction) resolvers* generate a voltage which is a specified function of the angular position of the rotor. Ordinarily, the stator and the rotor has each two coils at right angles to each other. Computing circuits use sine-cosine synchro resolvers, linear synchro resolvers, and scaling synchro resolvers.

Consider the operating principle of a sine-cosine synchro resolver whose circuit is shown in the diagram of Fig. 2.2a. A field winding, 1, energized with an alternating voltage,  $V_f$ , sets up a magnetic flux,  $\Phi$ , which cuts the rotor windings, 3 and 4. When the rotor takes up a position such that  $\alpha = 0$ , the emfs induced in the rotor windings will respectively be  $E_1 = 0$  and  $E_2 = E_{\max}$ . In the absence of loading, and with the rotor taking up

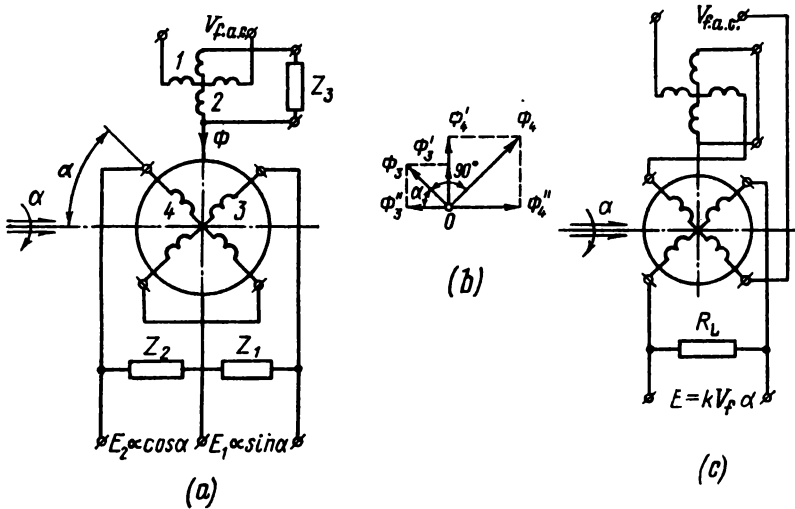


Fig. 2.2

a position such that  $\alpha \neq 0$ , the peak values of the emfs will be defined as

$$E_1 = E_{\max} \sin \alpha = m V_f \sin \alpha = (\omega_2 / \omega_1) V_f \sin \alpha \quad (2.12)$$

$$E_2 = E_{\max} \cos \alpha = m V_f \cos \alpha = (\omega_2 / \omega_1) V_f \cos \alpha \quad (2.13)$$

where  $m = \omega_2 / \omega_1$  is the turns ratio ( $\omega_2$  standing for the secondary and  $\omega_1$  for the primary turns; in sine-cosine synchro resolvers,  $m$  ranges from 0.1 to 1), and  $E_{\max}$  is the peak value of the emf induced in the rotor winding when it is at right angles to the flux  $\Phi$ .

With loads,  $z_1$  and  $z_2$ , connected across the output terminals, currents begin to flow in the rotor windings, giving rise to additional magnetic fluxes,  $\Phi_3$  and  $\Phi_4$ , which are at right angles to windings 3 and 4 (Fig. 2.2b). Resolving  $\Phi_3$  and  $\Phi_4$  along and across the rotor axis gives the total values of the longitudinal and transverse components

$$\Phi_l = \Phi_3' + \Phi_4'$$

$$\Phi_t = \Phi_3'' - \Phi_4''$$

Their interaction with the main magnetic flux  $\Phi$  in the field winding affects the output emfs differently.  $\Phi_f$  does not practically affect the output emfs because it is made good by an increase in the current through the field winding. In contrast,  $\Phi_i$ , on cutting the rotor windings, induces in them parasitic emfs which distort the waveforms of both  $E_1$  and  $E_2$ . A variety of methods are in use to compensate for the effect of  $\Phi_i$ , notably primary and secondary balancing. Then  $\Phi_f = \text{constant}$  and  $\Phi_i = 0$  for any value of  $\alpha$ .

Primary balancing consists in that a compensating winding, 2, is put on the stator at right angles to the field winding. Its impedance  $z_3$  is equal in magnitude to the internal resistance of the voltage source,  $V_f$ . Since the internal resistance of the source is small, the compensating winding is usually short-circuited. In sine-cosine resolvers with primary balancing,  $\Phi_i$  is balanced out by the magnetic flux set up by winding 2, which is equal in magnitude but opposite in sense to  $\Phi_i$ . With secondary balancing, the rotor windings are assumed to be identical and  $z_1 = z_2$ . Then  $\Phi'_3 = \Phi'_4$ ,  $\Phi_i = 0$ , and  $\Phi_f = \text{constant}$  for any value of  $\alpha$ . As often as not, primary and secondary balancing is combined to effect an almost complete compensation of  $\Phi_i$ .

Interconnection of the stator and rotor windings of one or several sine-cosine synchro resolvers produces computing circuits to perform linear transformations, coordinate conversion, multiplication and some other operations.

The circuit of a linear synchro resolver with primary balancing is shown in Fig. 2.2c, for which

$$E = V_f \frac{m \sin \alpha}{1 + m \cos \alpha} \quad (2.14)$$

where  $m = \omega_2/\omega_1$  as before (for linear synchro resolvers,  $m = 0.565$ ).

In the range of  $\alpha = \pm 60^\circ$ , the relationship is linear very nearly

$$E = kV_f\alpha$$

where  $k$  is a proportionality factor ( $k = 0.0635 \text{ deg}^{-1} = 0.364 \text{ rad}^{-1}$ ).

Holding the rotor of a sine-cosine synchro resolver stalled at some value of  $\alpha$  produces a scaling synchro resolver, because with  $\alpha$  held constant,  $E_1 = k_1V_f$  and  $E_2 = k_2V_f$ , where  $k_1$  and  $k_2$  are constant.

The circuit shown in Fig. 2.3a may be used to convert the rectangular (Cartesian) coordinates  $x$  and  $y$  into polar coordinates,  $\rho$  and  $\theta$  (that is, to compute a vector from its rectangular components). The equations governing the conversion are

$$\begin{aligned} \rho &= \sqrt{x^2 + y^2} \\ \theta &= \arctan(y/x) \end{aligned}$$



Energizing the stator windings with voltages  $V_x \propto x$  and  $V_y \propto y$  sets up a total flux (Fig. 2.3b) defined as

$$\Phi = \sqrt{\Phi_x^2 + \Phi_y^2} = k\sqrt{V_x^2 + V_y^2}$$

where  $\Phi_x = kV_x$  and  $\Phi_y = kV_y$  are the mutually perpendicular magnetic fluxes produced by windings 2 and 1.

At  $\alpha \neq \varphi$ , that is, when rotor winding 4 is not aligned with the direction of the flux  $\Phi$ , an error signal,  $\Delta V$ , is generated, which actuates the associated servo system. The servo motor then moves

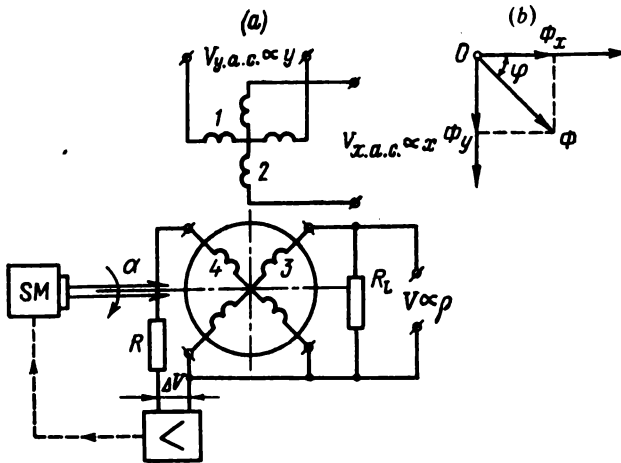


Fig. 2.3

the rotor into a position where  $\alpha = \varphi$  and  $\Delta V = 0$ . Then winding 4 will be parallel with, and winding 3 perpendicular to, the flux  $\Phi$  and

$$V = k_1 \Phi = k k_1 \sqrt{V_x^2 + V_y^2} \quad (2.15)$$

$$\alpha = \varphi = \arctan (V_y / V_x) \quad (2.16)$$

From comparison of Eqs. (2.15) and (2.16) with the expressions for  $\rho$  and  $\theta$ , it can readily be noted that, when energized with  $V_x \propto x$  and  $V_y \propto y$ , the above circuit generates  $V \propto \rho$  and  $\alpha = \theta$ .

Conversely, from comparison of Eqs. (2.12) and (2.13) with the expressions for  $x$  ( $x = \rho \cos \theta$ ) and  $y$  ( $y = \rho \sin \theta$ ), it is an easy matter to see that an ordinary sine-cosine synchro resolver (see Fig. 2.2a) may be used to obtain the polar to rectangular transformation. Then,  $E_1$  will be proportional to  $y$  and  $E_2$  to  $x$ , if  $E_{\max}$  is proportional to  $\rho$  and  $\alpha = \theta$ .

**2.2.3. Tachometer generators.** A tachometer (or rate) generator is a precision electromechanical component resembling a small motor and having an output voltage proportional to the rotational speed of its shaft. Rate generators have found extensive use as basic computing elements in electromechanical differential analyzers, integrators, and similar machines.

Tachometer generators may be d.c. or a.c., according to the output emf generated. D.c. rate generators may be separately excited or permanent-magnet excited. In each case they have a commutator. The field winding of a separately excited rate

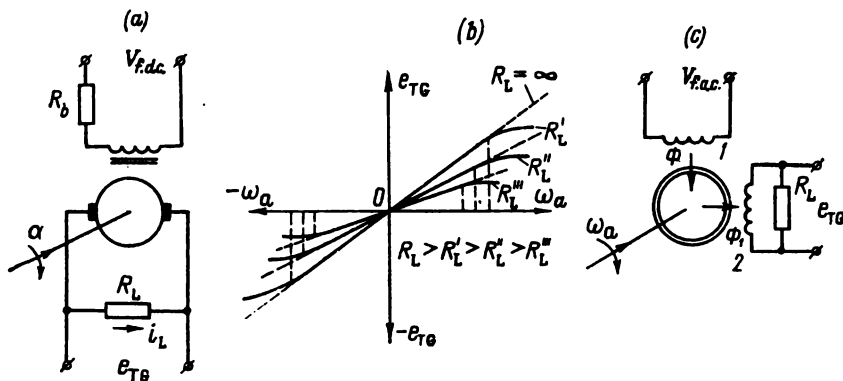


Fig. 2.4

generator (Fig. 2.4a) is energized with a direct voltage,  $V_f$ . As the armature revolves in the magnetic field due to the field winding at an angular speed  $\omega_a$ , an emf is induced in the armature winding, defined (in the absence of loading) as

$$e_{TG} = k_{TG} V_f \omega_a = k_{TG} V_f (d\alpha/dt) \quad (2.17)$$

where  $k_{TG}$  = proportionality factor (the tachometer generator constant)

$\alpha$  = angular position of the armature relative to its reference position

From Eq. (2.17) it follows that a tachometer generator can perform differentiation.

In the absence of loading, the response of a tachometer generator,  $e_{TG} = f(\omega_a)$ , is linear (Fig. 2.4b). With the load resistance,  $R_L$ , connected across the output terminals of the rate generator, the response ceases to be linear when  $\omega_a$  exceeds a certain range of values. This is because, on traversing the armature circuit, the load current  $i_L$  gives rise to an armature-reaction magnetic flux defined as

$$\Phi_{a.r} = k i_L \omega_a$$

which interacts with the main excitation flux to reduce it in magnitude and to distort the linear response. The greater the load resistance,  $R_L$ , the wider the range over which the response is linear. Several methods are used to reduce armature reaction, such as an increase in  $R_L$ , an increase in the air gap between the armature and stator, putting a series compensating winding on the stator, etc.

Temperature compensation (that is, compensation for the complementary error due to changes in the resistance of the stator winding with variations in temperature) is provided by placing a ballast resistor,  $R_b$ , in series with the field winding, with a value considerably exceeding that of the latter. Then the current in the excitation circuit and, as a consequence, the magnetic flux will mainly be determined by the value of  $R_b$  and will be less dependent on temperature variations.

Proper selection of materials for magnetic-circuit punchings, commutator bars and brushes goes a long way towards improving the accuracy of rate generators. The ripple in  $e_{TG}$  due to the commutator can be smoothened by placing a filter in the armature circuit and increasing the number of sections (at least to 20-25) in the winding.

D.c. tachometer generators with permanent-magnet excitation have not practically found use in computing circuits because of instabilities in their magnetic characteristics. Yet, they are often employed in servo systems as regenerative or degenerative feedback elements.

Because they use no commutator, a.c. tachometer generators are much simpler to make and are more reliable in use. They may be of any one of two types: synchronous and induction. In synchronous tachometer generators, the armature is a permanent magnet with several pairs of poles, and the output winding is dropped in slots in the stator. Unfortunately, they suffer from several drawbacks (the frequency of the output emf is dependent on the angular speed of the armature, the phase is independent of the sense of armature rotation), which limit their utility.

In contrast, induction tachometer generators are widely used in various computing circuits. An induction tachometer generator (Fig. 2.4c) has two mutually perpendicular windings in the stator (field winding 1 energized with a single-phase alternating voltage,  $V_f$ , and output winding 2) and a short-circuited rotor, or armature, which is usually fabricated as a thin-walled aluminium or phosphor-bronze cup (hence the name 'drag-cup' rotor) to reduce the moment of inertia and the ripple in the output emf. The field winding gives rise to a pulsating magnetic flux,  $\Phi$ . At  $\omega_a = 0$ ,  $e_{TG} = 0$ , because winding 2 is parallel with the direction of the flux  $\Phi$ . When the armature revolves in the magnetic field,

currents are induced in it, forming an alternating magnetic flux,  $\Phi_1$ , at right angles to the flux  $\Phi$ . As a result, a sinusoidal emf is induced in the output winding, proportional to  $\omega_a$ :

$$e_{TG} = -k\omega_a$$

Its frequency is equal to that of the excitation voltage, and its phase undergoes a reversal each time the direction of armature rotation is reversed. With an appropriate selection of materials and careful manufacture, the response can be maintained linear over a sufficiently wide range and the accuracy can be kept to within 0.1%.

**2.2.4. Operational amplifiers.** These are high-gain direct-coupled (d. c.) amplifiers with feedback. They are widely used in electronic analog computers to perform the basic linear operations of multiplication by a constant (scaling), summation, sign inversion, integration and the more complex operations.

Among the special requirements that an operational amplifier as an element of an analog computer should meet are the following: it should have an odd number of stages to simplify negative feedback, a high gain without feedback (40,000 to 100,000 or more), a high input and a low output impedance, a negligible d. c. drift, a sufficiently wide bandwidth, an unbalanced circuit configuration to simplify the cascading of several operational amplifiers, and an output voltage essentially linear within the range of interest. As a rule, operational amplifiers have three stages, use a large amount of negative voltage feedback, and operate on stabilized power supplies.

### Operational Amplifier as a Computing Device

A generalized block diagram of an operational amplifier complete with input and feedback circuit components appears in

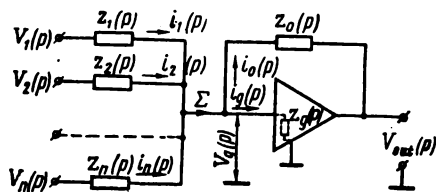


Fig. 2.5

Fig. 2.5. Referring to the diagram, the  $V_i(p)$ 's are input voltages (where  $i = 1, 2, \dots, n$ ), the  $i_i(p)$ 's and  $i_o(p)$ 's are the currents in the input and feedback circuits, respectively, the  $z_i(p)$ 's are the input impedances, the  $z_o(p)$  and  $z_g(p)$  are the feedback impedance

and the grid-circuit impedance, respectively, and  $k$  is the gain of the amplifier without feedback. At high values of  $k$  and  $z_g$  and a limited value of  $V_{out}$ , the grid current,  $i_g(p)$ , may practically be neglected, that is,  $i_g(p) \approx 0$ . Then the grid voltage,  $V_g(p)$ , at the

summing junction,  $\Sigma$ , will likewise be zero very nearly, because of which the summing junction is said to be at virtual earth potential. In operational notation, the currents for the summing point may be described as

$$i_1(p) + i_2(p) + \dots + i_n(p) = i_0(p) + i_g(p) \approx i_0(p) \quad (2.18)$$

In terms of voltages and impedances, the currents may be written as

$$\begin{aligned} \frac{V_1(p) - V_g(p)}{z_1(p)} + \frac{V_2(p) - V_g(p)}{z_2(p)} + \dots + \frac{V_n(p) - V_g(p)}{z_n(p)} \\ = \frac{V_g(p) - V_{out}(p)}{z_0(p)} \end{aligned} \quad (2.19)$$

Substituting

$$V_g(p) = -V_{out}(p)/k \quad (2.20)$$

in Eq. (2.19) gives

$$\begin{aligned} \frac{V_1(p)}{z_1(p)} + \frac{V_2(p)}{z_2(p)} + \dots + \frac{V_n(p)}{z_n(p)} + V_{out}(p) \left\{ \frac{1}{k} \left[ \frac{1}{z_1(p)} \right. \right. \\ \left. \left. + \frac{1}{z_2(p)} + \dots + \frac{1}{z_n(p)} + \frac{1}{z_0(p)} \right] + \frac{1}{z_0(p)} \right\} = 0 \end{aligned} \quad (2.21)$$

At high values of  $k$ ,

$$\frac{1}{k} \left[ \frac{1}{z_1(p)} + \frac{1}{z_2(p)} + \dots + \frac{1}{z_n(p)} + \frac{1}{z_0(p)} \right] \approx 0$$

and

$$\begin{aligned} V_{out}(p) = -\frac{z_0(p)}{z_1(p)} V_1(p) - \frac{z_0(p)}{z_2(p)} V_2(p) - \dots - \frac{z_0(p)}{z_n(p)} V_n(p) \\ = -\sum_{i=1}^n \frac{z_0(p)}{z_i(p)} V_i(p) \end{aligned} \quad (2.22)$$

It is seen from Eq. (2.22) that at high values of  $k$  the output voltage and, as a consequence, the accuracy of the operational amplifier is independent of its circuit parameters and is solely decided by the precision and stability of the impedances,  $z_i(p)$  and  $z_0(p)$ .

The ratio

$$w(p) = z_0(p)/z_i(p)$$

is called the transfer function of the operational amplifier with respect to the  $i$ th input.

The operational amplifier may perform various mathematical operations, according as  $z_0(p)$  and  $z_i(p)$  are resistive, capacitive or inductive impedances and according to the connections of the amplifier components. Some of the most important applications of the operational amplifier are discussed below.

(1) The operational amplifier as a summer. With  $n$  inputs and with pure resistances in the input and feedback circuits, we have

$$z_i p = R_i, \quad z_0(p) = R_0$$

Then

$$V_{out} = - \sum_{i=1}^n \frac{R_0}{R_i} V_i \quad (2.23)$$

As is seen, the output is the negative sum of the input voltages, each multiplied by a constant which is the ratio of the feedback resistance to the particular input resistance. At  $R_1 = R_2 = \dots = R_n = R$ ,

$$V_{out} = - \frac{R_0}{R} \sum_{i=1}^n V_i \quad (2.24)$$

while at  $R = R_0$

$$V_{out} = - \sum_{i=1}^n V_i \quad (2.25)$$

(2) The operational amplifier as a scaler. Assuming that  $n = 1$ ,  $z_1(p) = R_1$ ,  $z_0(p) = R_0$  and  $V_1(p) = V_{in}$ , we have

$$V_{out} = - \frac{R_0}{R_1} V_{in} = a V_{in} \quad (2.26)$$

that is, the operational amplifier multiplies the input variable by a constant, an operation for which in this context it is convenient to use a special name, scaling, and the unit which does this is called a scaler.

(3) The operational amplifier as a differentiator. With  $n = 1$  and by using a capacitor  $C$  as the input impedance and a resistor  $R_0$  as the feedback impedance, we have

$$z_1(p) = \frac{1}{pC}, \quad z_0(p) = R_0, \quad V_1(p) = V_{in}(p)$$

whence

$$V_{out}(p) = - \frac{R_0}{1/pC} V_{in}(p) = - p R_0 C V_{in}(p)$$

and, on recovering the original time function,

$$V_{out} = - R_0 C \frac{dV_{in}}{dt} \quad (2.27)$$

(4) The operational amplifier as an integrator and a summing integrator. Given  $n = 1$ ,  $V_1(p) = V_{in}(p)$ ,  $z_1(p) = R_1$ , and  $z_0(p) = 1/pC$ , we get

$$V_{out}(p) = - \frac{1}{p R_1 C} V_{in}(p)$$

Or, on recovering the original time function,

$$V_{out} = - \frac{1}{R_1 C} \int_0^t V_{in} dt \quad (2.28)$$

With  $n$  inputs,  $z_i(p) = R_i$  and  $z_0(p) = 1/pC$ ,

$$V_{out}(p) = - \frac{1}{p} \sum_{i=1}^n \frac{1}{R_i C} V_i(p)$$

and, on recovering the original time function,

$$V_{out} = - \int_0^t \sum_{i=1}^n \frac{1}{R_i C} V_i dt \quad (2.29)$$

that is, the operational amplifier integrates a sum of input voltages,  $V_i$ .

According to Eqs. (2.24), (2.27) and (2.28), the static gain of an operational amplifier connected as a summer, a differentiator and an integrator respectively is:  $k_{st} = R_0/R$  (with respect to the  $i$ th input),  $k_d = R_0 C$ ,  $k_{int} = 1/R_1 C$ .

An operational amplifier may be adapted to perform some other operations and represent linear and nonlinear functional relationships by choice of resistors, capacitors, and inductors and their combinations as input and output impedances. Some of the most commonly used computing circuits built around an operational amplifier are listed in Table 2.1.

### Valve Operational Amplifiers

The circuit of a widely used three-stage operational amplifier with parametric drift compensation is shown in Fig. 2.6. A major drawback of this type of amplifier is drift, that is, slow variations with time in output voltage for zero input voltage. It is mainly caused by instabilities in circuit parameters, variations in supply (anode, bias and filament) voltages, degradation of cathode emission, and grid currents. Since these variations are rather slow, the drift voltage is a low-frequency component.

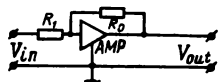
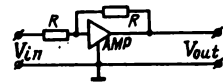
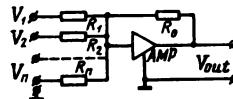
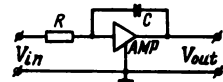
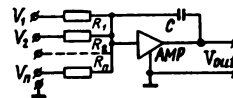
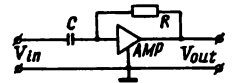
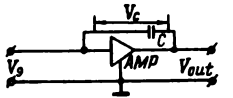
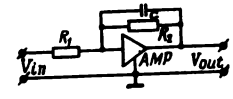
For purposes of analysis, drift is usually referred to input. Then the drift emf can be defined as

$$e_d = V_d/k$$

where  $V_d$  is the drift component in output voltage.

The magnitude of drift is most of all affected by the first stage. A number of special circuit configurations have been developed for the first stage to compensate for drift by suitable choice of its parameters (hence, the name 'parametric compensation'). Most

Table 2.1

Nos.	Computing element	Relation	Circuit
1	Scaler	$V = -\frac{R_0}{R_1} V_{in}$	
2	Sign inverter	$V_{out} = -V_{in}$	
3	Summer	$V_{out} = -\sum_{i=1}^n \frac{R_0}{R_i} V_i$	
4	Integrator	$V_{out} = -\frac{1}{RC} \int_0^t V_{in} dt$	
5	Summing integrator	$V_{out} = -\int_0^t \sum_{i=1}^n \frac{1}{R_i C} V_i dt$	
6	Differentiator	$V_{out} = -RC \frac{dV_{in}}{dt}$	
7	Storage element	$V_{out} = \frac{k}{1+k} V_C$	
8	Representation of a lag element	$V_{out}(p) = -\frac{R_2}{R_1(1+pR_2C)} V_{in}(p)$	



often, resort is made to cathode drift compensation (see the first stage in the diagram of Fig. 2.6). The left-hand section of a dual triode provides amplification, and the right-hand section compensates for drift caused by variations in filament voltage and cathode emission current. These factors may be replaced with an equivalent voltage source,  $e_d$ , in the cathode circuit. The objective of compensation is to hold the stage output voltage,  $V_{out}$ , constant. The condition for complete drift compensation can then be written as

$$\Delta i_{a1} = 0, \quad i_{a1} = \text{constant} \quad (2.30)$$

Suppose,  $e_d$  increases. This will bring about an increase in cathode current,  $i_k$ , and also in  $i_{a1}$  and  $i_{a2}$ , and also in the voltage drop

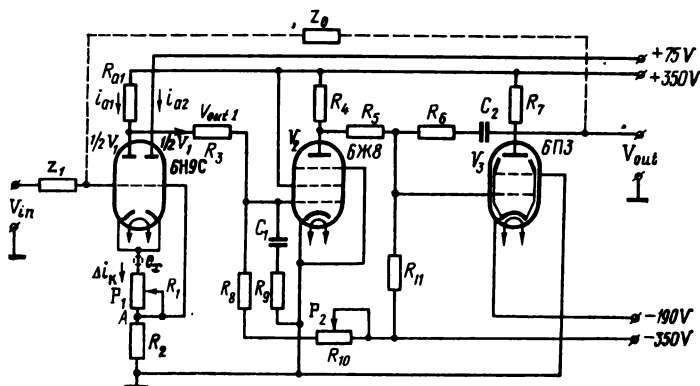


Fig. 2.6

across the cathode resistance,  $R_k = R_1 + R_2$  and across  $R_1$ . As a result, the potential at point A connected to the grid of valve  $V_2$  with respect to the cathode will decrease, and so will the current  $i_{a2}$  by an amount  $\Delta i_{a2}$ . By suitable selection of the circuit parameters one obtains

$$\Delta i_{a2} = -\Delta i_k$$

or, in words, the change in cathode current is fully compensated for. The condition, Eq. (2.30), is satisfied when

$$R_2 = R_{12}/\mu_2 = 1/g_{m2}$$

where  $\mu_2$  = amplification factor of the valve  $V_2$

$g_{m2}$  = transconductance of the same valve

The second amplifier stage uses a low-power pentode (such as a Soviet-made 6Ж8), and the third is built around a power beam tetrode (a 6П3). Output voltage is set to zero with a potentiometer,  $P_2$ . In contrast to the first two stages, the cathode

of the power beam tetrode is connected to a  $-190\text{V}$  source, so that an output voltage of both polarities may be obtained as the sign of  $V_{in}$  is reversed. The networks  $C1/R9$  and  $C2/R6$  stabilize amplifier operation.

The overall gain without feedback is 40,000 to 50,000. Drift is reduced to 1-3 mV over 10 min at unity gain and to 120 mV over 100 s when the amplifier is connected as an integrator with a time constant  $\tau = RC = 1$ . In the latter case, the integration time is kept down to practically not over 200 s with sufficient accuracy.

A much better way to reduce drift is to use what is known as chopper stabilization (i. e., to chop, that is, convert the d. c. signal

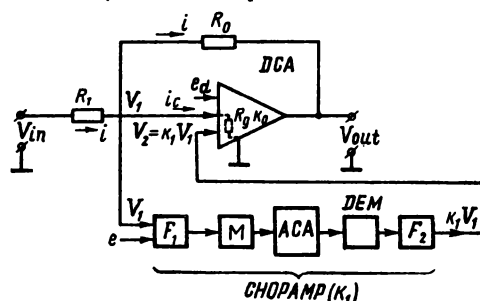


Fig. 2.7

to be amplified into a.c.). In such an arrangement (Fig. 2.7) there is an auxiliary a.c. amplifier inherently drift free. In a combination with filters,  $F1$  and  $F2$ , a chopper (acting as a modulator),  $M$ , and a rectifier (acting as a demodulator),  $DEM$ , the a.c. amplifier,  $ACA$ , forms an auxiliary driftless d.c. amplifier, often called a chopper amplifier (labelled  $CHOPAMP$  in the diagram). The combination of the main amplifier  $DCA$  and the chopper amplifier is usually spoken of as either a chopper-stabilized d.c. amplifier or a drift-corrected amplifier.

The a. f. input signal (together with the error),  $V_1$ , is applied to and boosted by both amplifiers,  $DCA$  and  $CHOPAMP$ . Of the two voltages,  $V_1$  and  $V_2$ , applied to the input of the main d.c. amplifier,  $V_2$  is amplified more, because  $V_2 = k_1 V_1 \gg V_1$ . In other words, it appears as if there is a series connection of two amplifiers,  $DCA$  and  $CHOPAMP$ , with an overall gain equal to

$$k = (1 + k_1) k_0 \approx k_1 k_0$$

As is seen, the chopper amplifier reduces the error in output voltage due to drift by a factor of  $(1 + k_1)$ . The symbol  $e$  in the diagram represents stray pick-up, hum, etc.

Should the frequency of the input signal rise to values falling outside the bandwidth of the chopper amplifier, its gain will drastically decrease to a point where the chopper amplifier actually ceases contributing to the performance of the entire circuit. Then the arrangement reduces to a conventional d.c. amplifier with  $k = k_0$ .

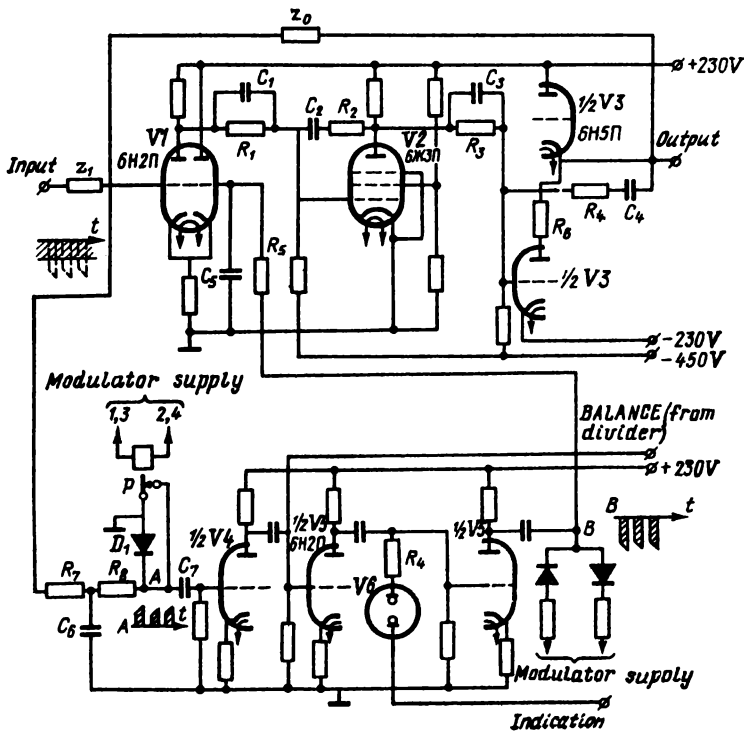


Fig. 2.8

As an example, Fig. 2.8 shows the circuit of a Soviet-made type Y-1 chopper-stabilized d.c. amplifier, used in the MH-14 analog computer. The input signal is applied to the grid of valve V1 in the first stage of the main d.c. amplifier and also, via a filter,  $R7/R8/C6$ , to the modulator (which is a polarized relay, with its armature returned to earth and its stationary contact connected to the input of the auxiliary a.c. amplifier) which converts the d.c. signal to be amplified to a.c. The chopped signal is then boosted by the auxiliary a.c. amplifier (using half the valve V4 and the valve V5), demodulated by a half-wave crystal-diode rectifier, or demodulator, smoothed by a filter,  $R5/C5$ , with a high

time constant, and applied to the second input of the valve  $V_1$  in the first stage of the d.c. amplifier connected as a summer. The output stage of the d.c. amplifier uses two series-connected triodes, which fact secures a considerable economy in power supply. The networks  $C_1/R_1$ ,  $C_2/R_2$ ,  $C_3/R_3$  and  $C_4/R_4$  provide the requisite signal shaping. The chopper amplifier, *CHOPAMP*, inverts the sign of the input signal owing to choice of phases for the supply voltages of the modulator,  $M$ , and the demodulator,  $DEM$ . The voltage applied to the polarized relay is limited by a silicon diode,  $D_1$ , so as to extend the service life of the relay contacts. Output voltage is set to zero by applying to the second stage of the chopper amplifier a voltage whose magnitude and phase can be set by a divider. Should a trouble or an abnormal condition occur in the chopper-stabilized d.c. amplifier, a neon lamp,  $NL$ , will illuminate, and a circuit will operate to automatically disable the operational amplifier.

The overall gain of a type  $V-1$  operational amplifier is  $k = 15 \times 10^3$  at 100 Hz ( $V_{in}$ ) and  $k = 10^3$  at 500 Hz for  $V_{out} = \pm 100$  V. At zero frequency,  $k_0 = 20 \times 10^3$  to  $25 \times 10^3$ , and  $k_1 = 2$  or  $3 \times 10^3$ . Drift is  $e_{d.in} = 0.2$  mV over 8 hours in operation as an amplifier and  $e_{d.in} = 10$  mV over 100 s in operation as an integrator for  $V_{in} = 0$  and  $RC = 1$  s. In integration, the time constant,  $\tau = RC$ , may be set anywhere between 0.01 and 10 s.

### *Transistor Operational Amplifiers*

Despite their limitations (variations in parameters with temperature, limited linear range of  $V_{out}$ , marked spread in parameters between units, low input impedance, low power output etc.), transistors have recently been gaining ground in circuits for computer elements, including operational amplifiers. Transistor circuits show longer service life, better reliability, low power drain, small size and weight, and freedom from filament or heater supplies.

The choice of a transistor amplifier circuit is to a marked degree determined by the requirements for accuracy and the type of the transfer function to be implemented. The fact that the transistor parameters are temperature-sensitive entails a considerable d.c. drift in transistor amplifiers and poses a number of difficulties in the synthesis of direct-coupled stages and input-output circuits.

As with valve amplifiers, transistor circuits widely use voltage feedback and conductive interstage coupling. An increase in input impedance is obtained by placing at the amplifier input an emitter follower which has a high input and a low output impedance. As often as not, the first amplifier stage uses compound-

connected transistors (Fig. 2.9). The transistors,  $T_1$  and  $T_2$ , are of the silicon junction type; they are less sensitive to temperature than germanium devices. In this circuit configuration the base current of  $T_2$  is the emitter current of  $T_1$ ; this arrangement results in a greater gain in comparison with the usual configuration.

D.c. drift in transistor amplifiers may be reduced in any one of several ways. Most commonly this is done by setting up the

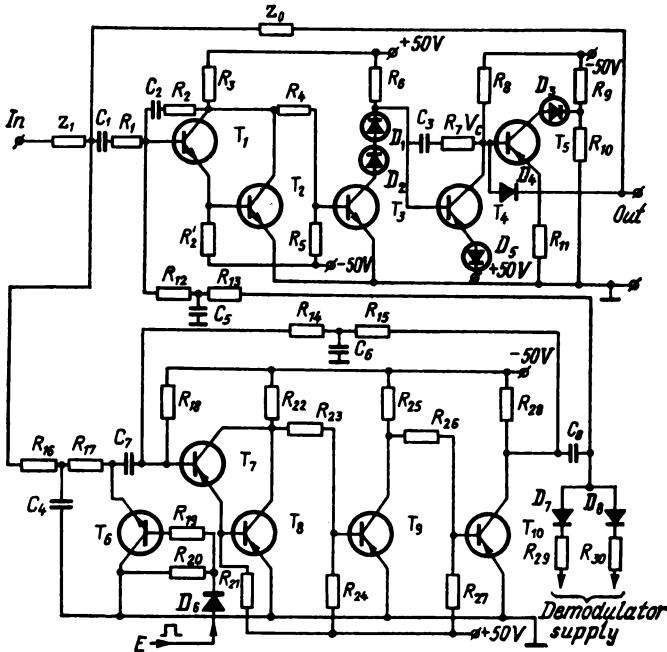


Fig. 2.9

first stage with two transistors having a common emitter load resistor (similar to cathode drift compensation used in valve amplifiers), series and shunt balancing circuits. Sometimes, the first stage of a transistor amplifier uses a valve, also as a way of reducing temperature-induced drift. The output stages must have a low output impedance but a considerable load current. The simplest form of an output stage might be an emitter follower. Unfortunately, it cannot secure a sufficient dynamic range for output voltage because the emitter-to-collector voltage is limited to a rather low value. As will be recalled, the dynamic range of an amplifier is the ratio of changes in output voltage  $V_{out\ max}$  to

drift  $V_d$  referred to input, or

$$d = V_{out\ max}/V_d$$

For valve amplifiers,  $d = 10^6$  to  $10^7$ ; for transistor amplifiers, such as the Soviet-made type УПТ-20М,  $d = 1.5 \times 10^4$  for  $V_{out\ max} = 30$  V and  $V_d = 2$  mV.

Also, the supply voltage of an emitter follower rises with increasing load. This is why an emitter follower is used as the output stage in applications where a broad bandwidth is not important and output load is low.

Any one of several circuits may be used to obtain high power output in combination with a broad bandwidth and a sufficient dynamic range; one of these output circuits is shown in Fig. 2.9. As is seen, the output stage uses *P-N-P* transistors,  $T_4$  and  $T_5$ . As long as the collector of  $T_4$  is held at a negative potential,  $V_C$ , diode  $D_4$  is in the OFF state. The emitter follower built around transistor  $T_5$  is conducting and operating only into the load resistor,  $R_{11}$ , which is at the same time the emitter bias resistor. No power is dissipated across  $R_{11}$ . Transistor  $T_4$  is OFF, and  $V_{out} < 0$ . As  $V_C$  rises above zero, diode  $D_4$  becomes conducting, the base and emitter of  $T_5$  are shorted together, and the transistor is driven to cut-off. However,  $T_4$  is turned ON, and  $V_{out} > 0$ . Thus, the transistors in the output stage operate in turn, as the input signal changes sign. This arrangement is often called an economy circuit because power dissipation is halved. The Zener diodes,  $D_3$  and  $D_5$ , in the emitter lead of  $T_4$  and the collector lead of  $T_5$ , respectively, drop part of supply voltage so that the power taken by the transistors will not exceed a preset level. Resistors  $R_9$  and  $R_{10}$  form a divider which fixes the voltage applied to the collector of  $T_5$ . Capacitor  $C_3$  and resistor  $R_7$  make up a local feedback circuit to prevent self-oscillation.

As a rule, transistor operational amplifiers are built as chopper-stabilized d.c. amplifiers as explained earlier, that is, with two parallel channels, an a.f. amplifier channel (incorporating a chopper amplifier (chopamp) and a d.c. amplifier. One such amplifier is shown in the diagram of Fig. 2.9. Designated УПТ-20М, it is used in the MH-10 computer. It is not unlike the Y-1 (or УПТ-10) valve amplifier. The first stage of the d.c. amplifier uses *N-P-N* transistors  $T_1$  and  $T_2$  which make up a compound-connected transistor. The transistors operate at a low collector voltage (about 1 V) and a low collector current (about 1 mA) due to the high value of the collector resistor,  $R_3$ , thereby minimizing drift in the first stage. Parasitic oscillation is prevented by local negative feedback (capacitor  $C_2$  and resistor  $R_2$ ). Interstage coupling uses both dividers and Zener diodes. This improves tem-

perature compensation without any loss in gain. The second stage uses a type П102 transistor, *T3*, connected in a conventional circuit which uses, however, Zener diodes, *D1* and *D2*, to stabilize the collector voltage. The second and third stages of the d. c. amplifier are direct-coupled. The third (output) stage is an emitter follower arranged into the economy circuit examined above; it delivers an output voltage of  $\pm 30$  V at an output current of 10 mA. When the output voltage is in positive polarity, transistor *T4* is conducting; when the output voltage is in negative polarity, transistor *T5* is conducting. The chopper amplifier (chopamp) incorporates a low-pass filter, *R16/R17/C4*, a transistor modulator, an a. c. amplifier, a demodulator, and a smoothing filter, *R12/R13/C5*.

The modulator circuit uses a silicon transistor, *T6*, connected in the reverse direction (that is, with the emitter and collector leads exchanged) to reduce drift and enhance accuracy of the modulator. The excitation (carrier) signal is applied to the base of *T6* via diode *D6* as a train of positive pulses following at a frequency of 500 Hz from a separate oscillator. Transistor *T6* is conducting when a positive pulse exists at its base, and is not conducting where there is no positive pulse at the base; that is, the circuit operates as a chopper. In comparison with an electromechanical chopper, the transistor circuit is more reliable, needs no frequent adjustments, and has a wider pass band for a. f. signals.

The a. c. amplifier is built around transistors *T7* through *T10* and uses potentiometers for interstage coupling. As in the d. c. amplifier, the first stage is a compound-connected transistor circuit based on transistors *T7* and *T8*. Strong negative direct-current feedback is placed around the chopper amplifier (chopamp) via resistors *R14* and *R15* and capacitor *C6*, in order to stabilize the operating conditions of the transistors, especially against temperature variations. The low-pass filter, *R15/C6*, eliminates negative feedback at the operating (carrier) frequency. As in the Y-1 amplifier (see Fig. 2.8), the demodulator is an ordinary half-wave rectifier circuit using diodes *D7* and *D8* and resistors *R29* and *R30*. Past the demodulator, the signal is smoothened by a filter *R13/C5*, and applied to the input of the d. c. amplifier. Since the d. c. amplifier and the a. c. amplifier have each three stages, their gain factors,  $k_0$  and  $k_1$ , are negative. The overall gain is likewise negative,  $k = k_0 k_1 < 0$ , because the gain of the demodulator is negative. This is accomplished by arranging for the transistor *T6* in the modulator and the diodes *D7* and *D8* in the demodulator to turn off during different half-cycles of the carrier — that is, the modulator and the demodulator operate in anti-phase. The gain of the YIT-20M operational amplifier is  $k = 5 \times 10^3$ ,

$8 \times 10^3$  and  $4 \times 10^6$  at 100 Hz, 10 Hz and d.c., respectively, at  $t = +35^\circ$  to  $-50^\circ \text{C}$ . Drift voltage is  $e_d = 3 \text{ mV}$  over the temperature range from 20 to  $50^\circ \text{C}$ . Power drain is 1.3 W.

## 2.3. Summing Networks

**2.3.1. General.** Summing networks (or summers) add together variables algebraically

$$z = \sum_{i=1}^n a_i x_i$$

or

$$z = k \sum_{i=1}^n x_i$$

where the  $a_i$ 's are the constant coefficients of the summation terms  $x_i$ 's, and  $k$  is a proportionality factor.

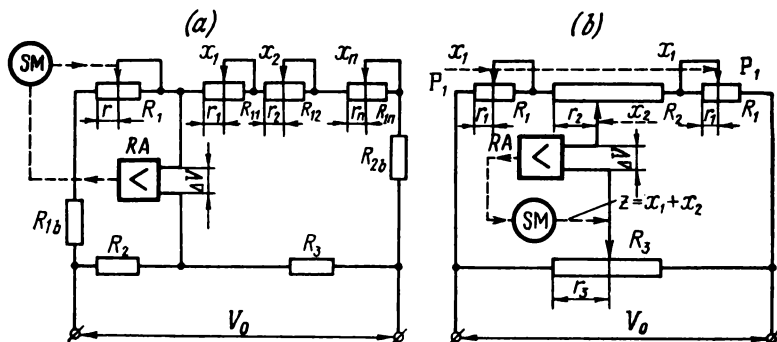


Fig. 2.10

Summing networks may be based on various physical principles.

**2.3.2. Bridge summing networks.** These may use either rheostats or potentiometers. In a rheostat summing network, the arm resistances may be changed independently of one another, while in the potentiometer type a change in one of the resistances brings about a corresponding change in the resistance of an adjacent arm.

A servo-operated rheostat summing network is shown in Fig. 2.10a. When the rheostat arms are set to their initial positions, for example  $r_1 = r_2 = \dots = r_n = 0$ , the circuit is at balance ( $\Delta V = 0$ ), described by an equation of the form

$$R_{1b}R_3 = R_{2b}R_2 \quad (2.31)$$



where  $R_{1b}$  and  $R_{2b}$  are the ballast resistors included to avoid a short circuit or a heavy current at low values of  $r$  and  $r_i$ . When the variables to be added together are set in as the displacements of the contact arms or resistances, the  $r_i$ 's, the balance of the bridge is upset, and an error signal,  $\Delta V$ , appears across the opposite pair of bridge junctions. This error signal activates the associated servo system incorporating a rotary amplifier,  $RA$ , and a servo motor,  $SM$ , which moves the contact arm of the rheostat,  $R_1$ , until the circuit takes up a new state of balance. The new state of balance may be described by an equation of the form

$$(r + R_{1b}) R_3 = [(r_1 + r_2 + \dots + r_n) + R_{2b}] R_2$$

whence, recalling Eq. (2.31), we get

$$r = (R_2/R_3) \sum_{i=1}^n r_i \quad (2.32)$$

Thus, for  $r_i \propto x_i$  we have

$$r \propto \sum_{i=1}^n x_i$$

If the origin be placed at the centre taps of the resistors  $R_{1b}$ , the circuit can be utilized to add together alternating quantities.

A potentiometer summing network is shown in Fig. 2.10b. As is seen, it has a twin potentiometer,  $P_1$ , the two halves of which are connected in adjacent arms of a bridge circuit. The wipers of the two halves are moved the same distance. After the variables,  $x_1$  and  $x_2$ , to be added are set in and the servo system has moved the wipers to a new state of balance, it can be described by an equation of the form

$$(r_1 + r_2)(R_3 - r_3) = [(R_2 - r_2) + (R_1 - r_1)] r_3$$

whence

$$r_3 = \frac{R_3}{R_1 + R_2} (r_1 + r_2) = k(r_1 + r_2) \quad (2.33)$$

If  $r_1 \propto x_1$  and  $r_2 \propto x_2$ , then

$$r_3 \propto z = x_1 + x_2$$

As compared with rheostat networks, the potentiometer type has a sensitivity twice as great.

Bridge summing networks may handle both a.c. and d.c. signals and show an accuracy of up to 0.05% of the range.

**2.3.3. Current summation.** Current summation is based on Kirchhoff's current law (KCL) for voltage sources connected in parallel. An elementary summing network for a number  $n$  of input voltages  $V_i$  is shown in Fig. 2.11a.

The current relation for point  $A$  is

$$i_0 = i_1 + i_2 + \dots + i_n \quad (2.34)$$

By expressing the currents in terms of voltages,  $V_i$ , input resistances,  $R_i$ , and the load resistance,  $R_L$ , we get after some manipulation

$$V_{out} = \sum_{i=1}^n \frac{1}{R_i \left( 1/R_L + \sum_{i=1}^n 1/R_i \right)} V_i = \sum_{i=1}^n k_i V_i$$

and for  $R_1 = R_2 = \dots = R_n = R$

$$V_{out} = \frac{1}{R/R_L + n} \sum_{i=1}^n V_i \quad (2.35)$$

The value of  $V_{out}$  is seen to be a function of the load resistance and the number of terms,  $n$ . These limitations may be avoided by

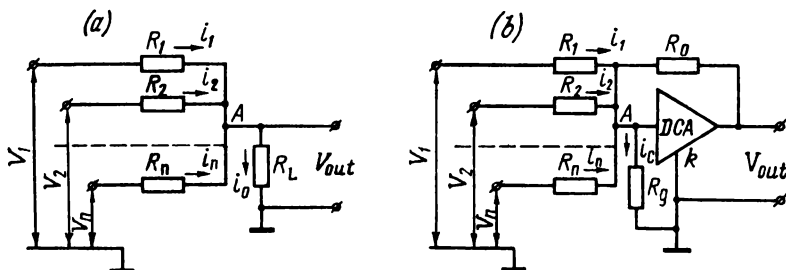


Fig. 2.11

placing an amplifier with strong negative feedback at the output (Fig. 2.11b). This circuit has been examined in Sec. 2.2 (see Fig. 2.5). The expression derived for this case, Eq. (2.23), contains neither  $R_L$  nor  $n$ , but the number of inputs to the summing amplifier is limited by the error. Putting in Eq. (2.21)  $V_i(p) = V_i$ ,  $z_i(p) = R_i$ , and  $z_0(p) = R_0$ , we get

$$\sum_{i=1}^n V_i/R_i + V_{out} \left[ \left( 1/k \sum_{i=1}^n 1/R_i + 1/R_0 \right) + 1/R_0 \right] = 0$$

whence

$$V_{out} = - \frac{\sum_{i=1}^n (R_0/R_i) V_i}{1/k \left( \sum_{i=1}^n R_0/R_i + 1 \right) + 1} \quad (2.36)$$

The fractional error is

$$\delta V_{out} = \frac{V_{out}^0 - V_{out}}{V_{out}^0} = 1 - V_{out}/V_{out}^0 \quad (2.37)$$

where  $V_{out}^0$  is defined by Eq. (2.36) and  $V_{out}$  by Eq. (2.23). Substituting (2.36) and (2.23) in (2.37), we obtain after simple manipulation

$$\delta V_{out} = -1/k \left( \sum_{i=1}^n R_0/R_i + 1 \right) \quad (2.38)$$

With  $\delta V_{out}$  specified in advance and for the selected values of  $R_0$  and  $R_i$ , it is possible to determine the value of  $k$  that will satisfy the specified accuracy. From Eq. (2.38) it is seen that  $\delta V_{out}$  is a function of the number of terms. For  $R_i = R$ , we have

$$\sum_{i=1}^n R_0/R_i = nR_0/R$$

This is the reason why in practical summing amplifiers it is usual to select  $n \leq 6$ . This circuit is widely used in analog computing elements.

**2.3.4. Voltage summation.** In contrast to current summation, the sources of emf in voltage summation are connected in series (Fig. 2.12). The voltages proportional to the variables to be added,  $V_i$  ( $i = 1, 2, \dots, n$ ), are set in with linear potentiometers,  $P_i$ , energized from independent sources of emf,  $E_i$ .

If the output of the circuit is coupled to a load resistance,  $R_L$ , which may, in the general case, be variable, the output voltage will be described as

$$V_{out} = iR_{eq} = \frac{V_1 + V_2 + \dots + V_n}{r_1 + r_2 + \dots + r_n + RR_L/(R + R_L)} \frac{RR_L}{R + R_L}$$

or

$$V_{out} = \frac{1}{\sum_{i=1}^n \frac{r_i}{R} (R/R_L + 1) + 1} \sum_{i=1}^n V_i = c \sum_{i=1}^n V_i \quad (2.39)$$

where  $R_{eq}$  is the equivalent resistance of the parallel connection of  $R$  and  $R_L$ , and  $c$  is a proportionality factor.

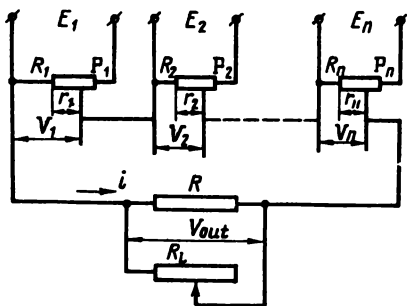


Fig. 2.12

To avoid the effect of  $r_i$  and  $R_L$  on  $V_{out}$ , their values must be selected to satisfy the condition such that

$$\sum_{i=1}^n r_{i \max} \ll R \text{ and } R \ll R_L$$

Then

$$V_{out} \approx \sum_{i=1}^n V_i$$

In adding together a.c. voltages, they are set in by transformers, in which case the frequency and phase of the unknown voltages must be the same.

## 2.4. Differentiators and Integrators

**2.4.1. General.** Differentiators and integrators produce an output proportional to the derivative or the integral of a function

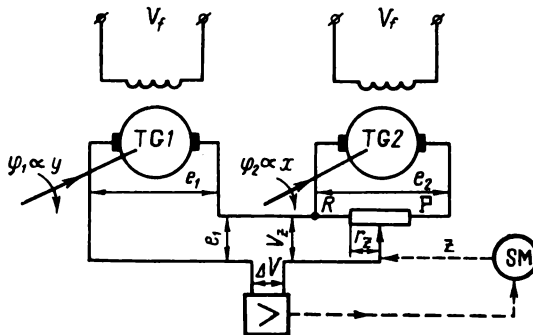


Fig. 2.13

with respect to time or any other independent variable, respectively. In mechanical differentiators and integrators the independent variable may be any. For electromechanical and electrical devices this is only time, although they may take a derivative or an integral with respect to an arbitrary argument, using some implicit computational techniques, such as the relation

$$z = dy/dx = \frac{dy/dt}{dx/dt} = y'/x' \quad (2.40)$$

which may be implemented by two differentiators with time as argument, and a divider.

Where the order of several derivatives has to be raised or lowered, resort is made to repeated differentiation and integration, using several differentiators or integrators connected in cascade.

By exchanging inputs and outputs, a differentiator may be converted to an integrator and back (the property of reciprocity).

**2.4.2. Differentiation and integration by means of tacho-generators.** With time as the independent variable, *differentiation* can be accomplished by a conventional tacho-generator circuit (see Sec. 2.2).

For differentiation of a function,  $y = f(x)$ , with respect to an arbitrary independent variable,  $x$ , use is made of an arrangement of two tacho-generators (Fig. 2.13). The function  $y$  and the argument  $x$  are set in as the angular positions,  $\varphi_1$  and  $\varphi_2$ , of the armatures of the tacho-generators,  $TG1$  and  $TG2$ , having identical characteristics. The voltage  $V_z$  taken from the wiper of a potentiometer,  $P$ , and the emf,  $e_1$ , are applied in opposition. At  $V_z \neq e_1$ , an error signal,  $\Delta V = V_z - e_1$ , exists, which causes the servo system to minimize the error. As a result, a state of equilibrium is reached in which

$$V_z = e_1$$

that is,

$$\Delta V = 0 \quad (2.41)$$

and the circuit is in a steady state. The emf's generated by the tacho-generators are defined by

$$e_1 = k_{TG} V_f d\varphi_1/dt \quad (2.42)$$

$$e_2 = k_{TG} V_f d\varphi_2/dt \quad (2.43)$$

where  $k_{TG}$  is the tacho-generator constant. The output voltage of the linear potentiometer is given by

$$V_z = (r_z/R) e_2 = \frac{k_{TG} V_f}{R} r_z d\varphi_2/dt \quad (2.44)$$

Substituting (2.44) and (2.42) in (2.41) and putting  $\varphi_1 = k_1 y$  and  $\varphi_2 = k_2 x$ , we get a relation from which  $r_z$  is defined as

$$r_z = (k_1 R/k_2) (dy/dx) = k dy/dx \quad (2.45)$$

that is, the derivative is reproduced as a resistance setting  $r_z$  of the potentiometer (the dial setting).

A function can be *integrated* with respect to time by the circuit of Fig. 2.14a. The function  $y = f(t)$  is set in as the voltage  $V = k_1 y(t)$ , taken from a functional generator. For steady-state conditions,

$$V = e_{TG}$$

or

$$\Delta V = V - e_{TG} = 0 \quad (2.46)$$

The output emf of the tacho-generator is given as

$$e_{TG} = k_{TG} V_f d\varphi/dt$$

Substituting the expressions for  $V$  and  $e_{TG}$  in Eq. (2.46), we obtain

$$k_1 y(t) = k_{TG} V_f d\varphi/dt$$

On integrating under zero initial conditions of  $\varphi$  and  $t$  and then determining the angle  $\varphi$ , we get

$$\varphi = \frac{k_1}{k_{TG} V_f} \int_0^t y(t) dt = k \int_0^t y(t) dt \quad (2.47)$$

that is, the integral is obtained as the angular position of the tacho-generator armature.

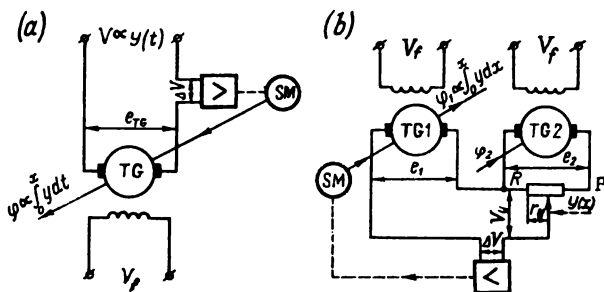


Fig. 2.14

So that a function can be integrated with respect to an arbitrary argument, the integrator should contain two tacho-generators (Fig. 2.14b). Then  $e_1$  and  $V_y$  will be defined as

$$e_1 = k_{TG} V_f d\varphi_1/dt$$

and

$$V_y = r_y e_2 / R = (r_y / R) k_{TG} V_f d\varphi_2 / dt$$

Substituting these expressions in the steady-state condition ( $V_y = e_1$ ) and putting  $r_y = k_1 y(x)$  and  $\varphi_2 = k_2 x$ , we get

$$\frac{k_1 k_2}{R} y(x) dx = d\varphi_1$$

By integrating this expression at  $\varphi_1^0 = 0$  and  $x^0 = 0$  we obtain

$$\varphi_1 = \frac{k_1 k_2}{R} \int_0^x y(x) dx = k \int_0^x y(x) dx \quad (2.48)$$

A major limitation of tacho-generator differentiators and integrators is that they include lag elements because of which they cannot be applied to fast-changing functions. Through a proper

selection of circuit parameters and the use of elements with a low time lag, the error may be reduced to 0.1%.

**2.4.3. Electrical and electronic differentiators and integrators.** Differentiation and integration utilize the transients occurring in passive  $RC$ -networks, that is, a capacitance and a resistance in series. In differentiation, the output is picked off the resistor; in integration this is done from the capacitor.

### *RC-networks*

For a differentiating  $RC$ -network (Fig. 2.15a), the relation between voltages in operational notation in the absence of loading

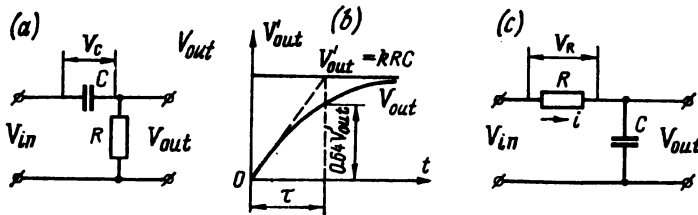


Fig. 2.15

may be described as

$$V_{in}(p) = V_C(p) + V_{out}(p)$$

On the basis of the equality

$$V_C(p) = (1/pC) i(p)$$

for zero initial conditions we get

$$V_{in}(p) = (1/pC) i(p) + i(p) R = i(p) (1/pC + R) \quad (2.49)$$

Multiplying (2.49) through by  $R$  and recalling that

$$i(p) R = V_{out}(p)$$

we obtain a differential equation describing the behaviour of an  $RC$ -network

$$pRCV_{out}(p) + V_{out}(p) = pRCV_{in}(p) \quad (2.50)$$

The first term in the above equation is an error. For a linear input voltage,  $V_{in}(t) = kt$ , we obtain a differential equation with a constant right-hand side

$$pRCV_{out}(p) + V_{out}(p) = kRC$$

the solution for which (in terms of time functions) has the form

$$V_{out} = RC (dV_{in}/dt) (1 - e^{-t/RC}) \quad (2.51)$$

Graphically, the plot of (2.51) is an exponential curve (Fig. 2.15b) which asymptotically approaches a horizontal straight line,  $V'_{out} = kRC$ . The time constant,  $\tau = RC$  (where  $R$  is in ohms,  $C$  in farads and  $\tau$  in seconds), describes the rate of the transients. The fractional error is

$$\delta V_{out} = \frac{V'_{out} - V_{out}}{V'_{out}} = e^{-t/RC} \quad (2.52)$$

Hence, the observation time,  $t_{ob}$ , at the end of which the error will not exceed a specified limit value,  $(\delta V_{out})_{lim}$ , is defined as

$$t_{ob} = RC \ln (\delta V_{out})_{lim} \quad (2.53)$$

As  $\tau = RC$  decreases,  $\delta V_{out}$  decreases too; that is, the circuit gives a faster response. At the same time, however,  $V_{out}$  decreases, and the accuracy is impaired. This is why in each particular case the circuit parameters  $R$  and  $C$  must be matched to strike a balance between these conflicting factors.

For an integrating  $RC$ -network (Fig. 2.15c) in the absence of loading and under zero initial conditions, we have

$$V_{in}(p) = V_R(p) + V_C(p) = i(p)R + (1/pC)i(p) = i(p)(R + 1/pC) \quad (2.54)$$

Since

$$i(p) = \frac{V_{in}(p) - V_{out}(p)}{R} \quad (2.55)$$

then, on substituting the expression for  $i(p)$  in (2.54), we obtain a differential equation

$$V_{out}(p) = (1/pRC)V_{in}(p) - (1/pRC)V_{out}(p) \quad (2.56)$$

where the first term on the right-hand side is the sought integral and the second is an error. From the equation the output voltage is found to be

$$V_{out}(p) = \frac{1}{1 + pRC} V_{in}(p) \quad (2.57)$$

Thus, the transfer function of an integrating  $RC$ -network is

$$w(p) = \frac{1}{1 + pRC}$$

At  $V_{in} = \text{constant}$ , the solution (in terms of original time functions) for Eq. (2.57) has the form

$$V_{out} = V_{in}(1 - e^{-t/\tau}) \quad (2.58)$$

By expanding the function  $\exp(-t/\tau)$  into a Maclaurin series and limiting ourselves to the expansion terms of at most order



two, we get

$$\exp(-t/\tau) = f(t) = f(0) + \frac{t}{1!} f'(0) + \frac{t^2}{2!} f''(0) = 1 - t/\tau + t^2/2\tau^2$$

Then,

$$V_{out} = V_{in}(t/\tau) - (V_{in}/2)(t^2/\tau^2)$$

where the first term indicates that the output voltage is linear, and the second term is an absolute error,  $\Delta V_{out}$ , which rises with time, thereby setting a limit to the maximum time of integration. This is why the circuit in question is used where the time of integration is small (for example, in pulse circuits) and no stringent requirements for accuracy exist.

### Differentiating and Integrating Operational Amplifiers

The passive  $RC$ -networks discussed above suffer from a number of drawbacks (low  $V_{out}$  in comparison with  $V_{in}$ , the effect of loading on accuracy, a limited time of integration) which preclude

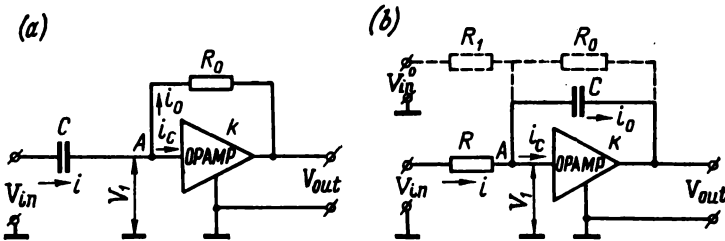


Fig. 2.16

their widespread use as computer elements. Stability and accuracy may be enhanced by providing an amplifier with strong feedback at the output of an  $RC$ -network, a combination (already examined in brief) known as an operational amplifier (often abbreviated to 'opamp').

In a differentiating opamp (Fig. 2.16a) there is a capacitor,  $C$ , at the input, and a resistor,  $R$ , in the feedback circuit. As has already been shown (see Sec. 2.2), the output voltage of such an opamp (for  $i_c = 0$  and a high gain,  $k$ ) is given by

$$V_{out} = -pR_0CV_{in}(p)$$

or in terms of original time functions

$$V_{out} = -R_0C dV_{in}/dt$$

A differentiating opamp, however, tends to accentuate the noise present in all electrical circuits, and it is therefore used but

seldom in analog computers. Yet, it is indispensable in some applications, such as where integration is performed with respect to an arbitrary variable  $x$ , implicitly, using the relation

$$\int_0^x f(x) dx = \int_0^t f(x) \frac{dx}{dt} dt$$

The integrator widely used in analog computers is the integrating opamp (shown by the solid lines in Fig. 2.16b). It deserves a closer examination.

Given a high input impedance and  $i_C \approx 0$ , we have

$$i(p) = i_0(p)$$

or

$$\frac{V_{in}(p) - V_1(p)}{R} = pC [V_1(p) - V_{out}(p)] \quad (2.59)$$

Substituting

$$V_1(p) = -V_{out}(p)/k$$

gives

$$V_{out}(p) [1 + pRC(k+1)] = -kV_{in}(p) \quad (2.60)$$

which is an equation describing a pure lag element with a gain  $k$  and a time constant  $\tau' = RC(k+1)$ .

The transfer function

$$w(p) = V_{out}(p)/V_{in}(p) = -\frac{k}{1 + pRC(k+1)} \quad (2.61)$$

may be rewritten as

$$V_{out}(p) = -\frac{1}{\frac{1}{k} + pRC\left(1 + \frac{1}{k}\right)} V_{in}(p)$$

Since  $k$  is high, it may be assumed that  $1/k \approx 0$ , whence

$$V_{out}(p) = -(1/pRC) V_{in}(p)$$

or, in terms of original time functions,

$$V_{out} = -1/RC \int_0^t V_{in} dt \quad (2.62)$$

Let us determine the fundamental error of integration. When the signal applied to the input of an integrating opamp is a step function,  $V_{in} = \text{constant}$ , the solution for Eq. (2.60) has the form

$$V_{out} = kV_{in} \{1 - \exp[-t/RC(k+1)]\} \approx kV_{in} [1 - \exp(-t/kRC)]$$

where  $k \gg 1$ . By expanding the function  $\exp(-t/kRC)$  into a Maclaurin series and discarding all expansion terms above the second order, we get

$$\exp(-t/kRC) = 1 - \frac{1}{kRC} t + \frac{1}{2(kRC)^2} t^2$$

Then,

$$V_{out} = \frac{V_{in}}{RC} t - \frac{V_{in}}{2k(RC)^2} t^2 \quad (2.63)$$

The second term in (2.63) is the absolute error of integration,  $\Delta V_{out}$ . The fractional error is

$$\delta V_{out} = \frac{\Delta V_{out}}{V_{out \max}} = \frac{1}{2kRC} t \quad (2.64)$$

where

$$V_{out \max} = \frac{V_{in}}{RC} t$$

From Eq. (2.64) it follows that the error increases with increasing time of integration. The error may be reduced by increasing the gain  $k$  and the time constant  $RC$ . Suppose, for example, that the percent error is specified to be  $\delta V_{out} = 0.2\%$  over the integration time  $t \leq 100$  s for  $\tau = RC = 1$  s. From Eq. (2.64), the desired value of  $k$  is found to be

$$k > k_{\min} = \frac{t}{2RC \delta V_{out \max}} = \frac{100}{2 \times 1 \times 0.002} = 25,000$$

It is usual to choose  $k = 40,000$  to  $50,000$ .

By rewriting Eq. (2.57) as

$$V_{out}(p)(1 + pRC) = V_{in}(p) \quad (2.65)$$

and comparing Eqs. (2.65) and (2.60) it is an easy matter to note that the time constant of an integrating opamp is  $(k+1)$  times that of a passive integrating  $RC$ -network. In other words, integration may be carried on for an appreciably longer time (practically,  $t_{\max} = 150$  to  $200$  s). On the other hand, the error rises at a rate which is  $1/(k+1)$  that of the error in an  $RC$ -network. The maximum time of integration is given by

$$t_{\max} = 2RC(k+1) \delta V_{out \lim}$$

Prior to integration, the initial conditions must be set as a voltage  $V_{out}^0$  at  $t=0$ . For this purpose, the resistor  $R$  is brought out of circuit and an auxiliary charging circuit containing  $R_1$  and  $R_0$  (shown by the dashed line in the diagram of Fig. 2.16b) is brought in. As a result, we obtain the circuit of an amplifier with pure lag, containing  $R_1$  at input and  $R_0$  and  $C$  in the feedback circuit. On setting  $V_{in}^0 = \text{constant}$  the output voltage will obey the exponential law

$$V_{out} = -\frac{R_0}{R_1} V_{in}^0 (1 - e^{-t/R_0C})$$

With time,  $\exp(-t/R_0C)$  tends to zero and  $V_{out}$  to

$$V_{out}^0 = -\frac{R_0}{R_1} V_{in}^0 = \text{constant}$$

Thus,  $V_{in}^0$  charges  $C$  to practically  $V_{out}^0$ , because this potential is applied to one plate of  $C$  while the other plate is returned to the summing junction  $A$  held at a virtually ground potential. Pressure on the "Start" button removes the auxiliary charging circuit,  $R$  is connected to the input, and the usual operation of integration is commenced. An integrating opamp can deliver a high output voltage mainly limited by the maximum allowable error.

An integrating opamp can integrate signals over a sufficiently wide range of frequencies. Owing to its high input and relatively low output impedance, an integrating opamp can readily be matched to other circuits.

## 2.5. Function Generators

**2.5.1. General.** Nonlinear functions of one or several independent variables are implemented by devices known as *function generators*. Consider function generators whose output is a function of one independent variable, most widely used in analog computers. In an electric function generator the output voltage proportional to a specified function  $y = f(x)$  is given by

$$V_{out} = F(V_{in})$$

Function generators may utilize widely differing physical principles. According to application, all function generators may be divided into two broad classes:

(1) general-purpose function generators which can handle a wide range of functions with only minor changes in the configuration of the same basic circuit;

(2) special-purpose function generators specifically designed to generate a particular function. The use of special-purpose function generators is warranted in cases where a desired function cannot be generated by a general-purpose function generator (for example, a characteristic with a considerable slope, such as backlash, gap, etc.) or a special-purpose function generator is simpler to synthesize.

A desired function may be implemented either with theoretical accuracy or approximately. In a function generator showing theoretical accuracy, errors are mainly determined by inaccuracies in the circuit components, wiring and assembly (function generators using induction resolvers, photoformers, functional potentiometers, etc.). In function generators with approximation, most widely used because of simplicity, reliability and economy, a function is reproduced by circuits utilizing various methods of approximation (among them are diode function generators, shaped-card poten-

tiometers, and the like). In most cases, they yield more accurate results over the range involved than the former type.

**2.5.2. Approximation of functions.** Functions may be approximated in any one of several ways. Those widely used in connection with function generators are piecewise-linear approximation, step approximation and curvilinear approximation. Of these methods, piecewise-linear approximation has found the most widespread use.

With piecewise-linear approximation of a function (Fig. 2.17a), the curve  $y = f(x)$  is approximated by a series of straight line segments  $y = \varphi(x)$  which touch the desired curve at several

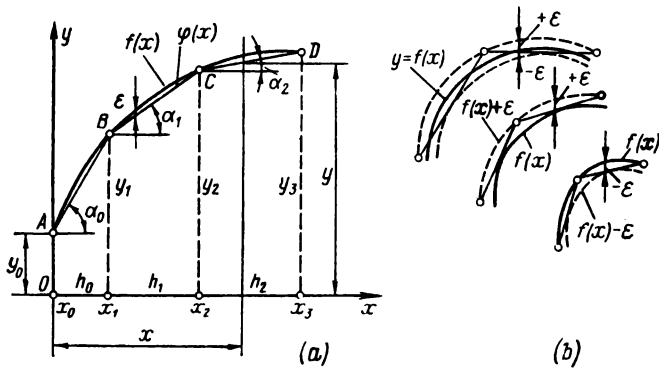


Fig. 2.17

points. For a function to be piecewise linearly approximable, it must satisfy the following conditions:

- (a) it must be univalued over the specified range of values for  $x$ ;
- (b) it must be continuous;
- (c) it must have a continuous first derivative.

Within each straight-line segment, the error of approximation must not exceed a predetermined limit,  $\epsilon$ .

The manner in which a convex curve can in most cases be approximated with straight-line segments is shown in Fig. 2.17b. The approximating segments are inscribed into a "tube" formed by the given curve and the parallel lines within the distance  $\epsilon$  of the curve. When the nodes of approximation are located on the desired curve  $f(x)$ , the length of an approximating segment is given by

$$h_k = \sqrt{8\epsilon/y''_{k \max}} \quad (2.66)$$

where  $y''_{k \max}$  is the maximum value of the second derivative of the function in the interval  $x_k \leq x \leq x_{k+1}$ .

Let the abscissae of approximation nodes be designated as  $x_0, x_1, \dots, x_n$ , the initial value for  $x = 0$  as  $y_0$ , the tilt of the straight-line segments to the axis of abscissae as  $\alpha_0, \alpha_1, \dots, \alpha_{n-1}$ . Then the ordinate within each segment will be changing as follows:

$$y = y_0 + (x - x_0) \tan \alpha_0, \quad x_0 \leq x \leq x_1$$

$$y = y_0 + (x_1 - x_0) \tan \alpha_0 + (x - x_1) \tan \alpha_1, \quad x_1 \leq x \leq x_2 \quad (2.67)$$

or

$$y = y_0 + \tan \alpha_0 (x - x_0) + (\tan \alpha_1 - \tan \alpha_0) (x - x_1)$$

if  $\tan \alpha_0 x$  is added to and subtracted from Eq. (2.67).

For the  $n$ th segment, the value of the function is given by

$$y = y_0 + \tan \alpha_0 (x - x_0) + (\tan \alpha_1 - \tan \alpha_0) (x - x_1) + (\tan \alpha_2 - \tan \alpha_1) (x - x_2) + \dots + (\tan \alpha_n - \tan \alpha_{n-1}) (x - x_n)$$

For  $x_0 = 0$ , the approximating broken line will be defined by an equation of the form

$$y = y_0 + kx + \sum_{i=1}^n b_i (x - x_i) \quad (2.68)$$

where  $k = \tan \alpha_0$ ,  $b_i = \tan \alpha_i - \tan \alpha_{i-1}$  ( $i = 1, 2, \dots, n$ )

$$b_i = \begin{cases} 0 & \text{for } x \leq x_{i \text{ init}} \\ \tan \alpha_i - \tan \alpha_{i-1} = \text{constant} & \text{for } x > x_{i \text{ in: t}} \end{cases}$$

where  $x_{i \text{ init}}$  is the value of  $x$  at the beginning of each segment.

Equation (2.68) gives the value of the approximating function for any value of  $x$ , if the nodes of approximation are chosen according to Eq. (2.66).

If a function to be approximated has been defined analytically, the second derivative  $y''_h$  can be found by differentiation. For this purpose, the entire range of values of the function is broken into intervals of monotonic variations in its second derivative, and within each interval approximation commences at a point corresponding to  $y''_{k \text{ max}}$ . Using the value of  $y''_{k \text{ max}}$  thus found and the limit of error,  $\epsilon$ , one then finds  $h_k$  from Eq. (2.66). Next,  $y''_{k-1}$  is found at the end of the segment and  $h_{k-1}$ , etc.

When a function is defined graphically or in tabular form, it can simply be approximated by a grapho-analytical method.

With step (piecewise-constant) approximation, a desired function is approximated by a staircase curve in which the straight-line segments of each step are parallel to the axes of coordinates.

**2.5.3. Tapped linear potentiometers.** In piecewise-linear approximation, a function of a single variable,  $y = f(x)$ , can be generat-

ed by a tapped linear potentiometer (Fig. 2.18). As a rule, the wiper of a tapped linear potentiometer is servo-driven across the arc of a circle.

To find the values for the padding resistors to be connected to the various taps on the potentiometer, we shall introduce the following notation:

$x_i$ 's ( $i = 1, 2, \dots, n$ ) are the abscissae of nodes  $b$ ,  $c$  and  $d$ ;

$r_i$ 's are the resistances of the padding resistors;

$R_i$ 's and  $R$  are the resistances of the padded segments and the total resistance of a linear potentiometer;

$r_{eq. i}$ 's are the resistances of the padders,  $r_i$ , connected in parallel with  $R_i$  within the respective segments;

$R_0$  is the fixed resistance corresponding to the function  $y = y_0 = f(x_0)$ ;

$l_i$ 's and  $l$  are the lengths of the individual segments and the total length of the potentiometer;

$\Delta V_i$ 's are the voltage drops across the potentiometer segments,  $l_i$ 's.

Let the function to be piecewise-linearly approximated have the form

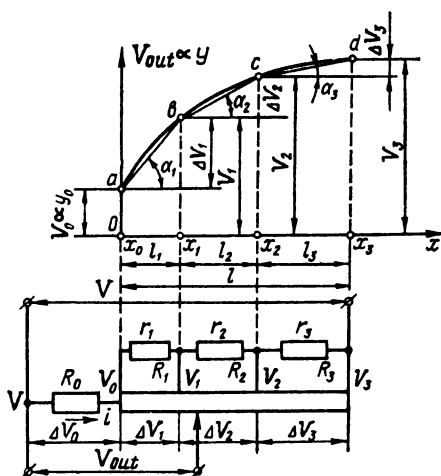


Fig. 2.18

$$V_{out} = f(x)$$

Then the voltage for each particular value of  $x$  will be defined as

$$V_{out} = y/M_y$$

where  $M_y$  is the scale factor. After approximation nodes are determined, the values of  $x_i$ ,  $l_i$ ,  $\Delta V_i = V_i - V_{i-1}$  and  $\Delta V_0 = V - V_0$  are found. The values of  $V$ ,  $R$ ,  $l$  and the limiting current,  $i_{lim}$ , are specified in advance or selected from design or other considerations. The values of  $R_i$  and  $R_0$  are given by

$$R_i = (l_i/l) R, \quad R_0 = \Delta V_0/i_{lim} \quad (2.69)$$

and the equivalent resistances,  $r_{eq. i}$ , are determined from

$$r_{eq. i} = r_i R_i / (r_i + R_i)$$

whence

$$r_i = r_{eq. i} R_i / (R_i - r_{eq. i}) \quad (2.70)$$

Since the current  $i$  is the same through all parts of the potentiometer, we may write

$$i = \Delta V_i / r_{eq.i} = V / r_{eq} \quad (2.71)$$

where  $r_{eq}$  is the total equivalent resistance of the tapped potentiometer given by

$$r_{eq} = \sum_{i=1}^n r_{eq.i}$$

and defined from considerations of the maximum power drain by

$$R_0 + r_{eq} = V^2 / P_{lim}$$

From Eq. (2.71), we have

$$r_{eq.i} = (r_{eq} / V) \Delta V_i \quad (2.72)$$

where  $r_{eq}$ ,  $V$  and  $\Delta V_i$  are as already defined. Substituting the expression for  $r_{eq.i}$  in Eq. (2.70) and knowing  $R_i$  from Eq. (2.69), we obtain the values of the padding resistors,  $r_i$ .

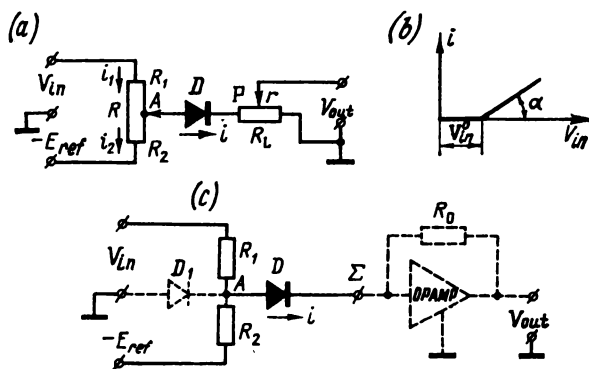


Fig. 2.19

Tapped linear potentiometers can approximate nonmonotonic functions of constant or alternating sign. As a first step, the function is broken down into intervals within which the function is monotonic (that is, the first derivative has a constant sign). As a rule, up to fifteen taps are provided. The total resistance is up to 30 kilohms. In designing a tapped-potentiometer function generator, allowance must be made for the loading error. If output load is considerable, an isolation amplifier will have to be provided. Tapped-potentiometer function generators are simple to align, draw little power, and secure sufficient accuracy (0.2%).

**2.5.4. Diode function generators.** These function generators widely used in analog computers utilize piecewise-linear approximation.



Diode function generators may use any one of several "diode packages", namely two-terminal, three-terminal, and three-terminal with a diode at virtual ground potential. The argument is set as an input voltage,  $V_{in}$ . For the purpose of analysis, it is assumed that the diode resistance is zero very nearly,  $R_d \approx 0$ .

Two-terminal diode packages have not found practical use because they require a floating source of voltage reference,  $E_{ref}$ , which fact complicates their connection into circuit.

A three-terminal diode package (Fig. 2.19a), ordinarily used in general-purpose diode function generators, includes a voltage divider with two inputs ( $V_{in}$  and  $E_{ref}$ ). The setting of the divider locates the breakpoint ( $V_{in} = V_{in}^0$ ), that is the instant when the diode is rendered conducting or non-conducting, while the setting of a potentiometer,  $P$ , controls the slope of the conduction line,  $\alpha$ . For  $E_{ref} < 0$  and  $V_{in} = 0$ , we find that  $V_A < 0$ , that is, the diode is OFF. As  $V_{in}$  is raised,  $V_A$  rises but remains negative as far as the breakpoint where  $V_{in} = V_{in}^0 = (R_1/R_2) E_{ref}$  for which  $V_A = 0$ . At  $V_{in} > V_{in}^0$ , the current through the diode is  $i = i_1 - i_2$  (with  $R_d = 0$ ). By replacing currents with voltages, we obtain

$$V_A/R_L = \frac{V_{in} - V_A}{R_1} - \frac{V_A + E_{ref}}{R_2}$$

whence

$$V_A = \frac{V_{in} - (R_1/R_2) E_{ref}}{R_1(1/R_2 + 1/R_L) + 1}$$

The output voltage will be

$$V_{out} = \beta V_A = \frac{\beta [V_{in} - (R_1/R_2) E_{ref}]}{R_1(1/R_2 + 1/R_L) + 1} \quad (2.73)$$

where  $\beta = r/R_L$ .

The slope of the diode characteristic (Fig. 2.19b) is defined as

$$\alpha = \tan^{-1} \frac{V_{out}}{V_{in} - V_{in}^0} = \frac{\beta}{R_1(1/R_2 + 1/R_L) + 1}$$

If  $R_1 = 0$  and  $r = R_L$  (that is,  $\beta = 1$ ), the slope,  $\alpha$ , is a maximum

$$\alpha_{max} = \tan^{-1} \beta = 45^\circ$$

As is seen, the slope of a three-terminal diode package is limited, which is a serious disadvantage.

In a package with a diode returned to virtual ground potential (Fig. 2.19c), there is no load resistor. The diode cathode is coupled directly to an amplifier at the summing junction (at virtual ground potential),  $\Sigma$ . Both the slope,  $\alpha$ , and the location of the breakpoint,  $V_{in}^0$ , are adjusted with the same divider,  $R_1/R_2$ . As

long as  $V_{in} \leq V_{in}^0$ , the diode is OFF, but it is rendered conducting as soon as  $V_{in}$  exceeds  $V_{in}^0$ , when  $V_A = 0$ . At that instant, the point  $A$  is returned to virtual ground potential via the diode, and the circuit reduces to a conventional summing amplifier with two inputs  $V_{in}$  and  $-E_{ref}$ . If  $V_{in}$  keeps rising, the potential at point  $A$  will not change, remaining equal to  $V_A = V_z$ . The output voltage of the amplifier is given as

$$V_{out} = - \left( \frac{R_0}{R_1} V_{in} - \frac{R_0}{R_2} E \right) \quad (2.74)$$

By differentiating it with respect to  $V_{in}$  and discarding the “—” sign, we get

$$dV_{out}/dV_{in} = \tan \alpha = R_0/R_1$$

whence the slope of the diode characteristics is

$$\alpha = \tan^{-1} R_0/R_1$$

As  $R_0 \rightarrow \infty$ , the slope is close to  $\pi/2$ . Because of this, functions with a considerable slope can be approximated by this diode circuit. The circuit with a diode returned to virtual ground potential is simpler, but it is less flexible because it has no separate adjustments for the breakpoint and the slope. This is why such a function generator is ordinarily used to generate functions not involving adjustments.

The diodes in the above circuits may be valves or semiconductor devices. The reverse resistance of a crystal diode is a finite quantity, and so when a negative potential difference is applied to the diode, a reverse current of a few tens of microamperes flows through it. To minimize the error due to the reverse current, it is customary to place an additional diode,  $D_1$ , in the function generator circuit.

Any one of two operating modes may be chosen for a diode function generator, namely:

(a) mode  $A$  (Fig. 2.20  $a$  and  $b$ ), where the diode remains OFF as long as  $V_{in} = 0$  and turns on when  $V_{in}$  has risen to a predetermined value,  $V_{in}^0$ . This circuit configuration is employed to approximate functions whose first derivative is decreasing in absolute value;

(b) mode  $B$  (Fig. 2.20  $c$  and  $d$ ), where the diode is conducting as long as  $V_{in} = 0$  and turns off when  $V_{in}$  has risen to a preset value,  $V_{in}^0$ . This arrangement is employed in cases where the first derivative of the function is increasing in magnitude.

In Fig. 2.20, the values of  $V_{in}$  and  $E_{ref}$  in parentheses apply to a diode connected in the reverse direction (shown by the dotted lines in the figure). In the plots, they are represented by the dotted segments.

If the function to be approximated is nonmonotonic, both circuit configurations can be used in a diode function generator.

All diode function generators may be divided into general-purpose and special-purpose.

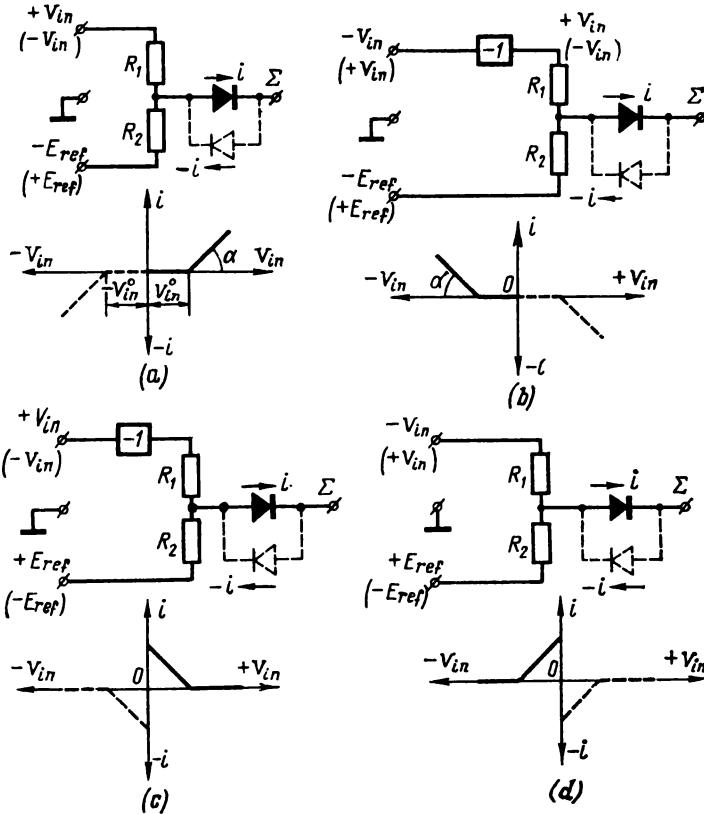


Fig. 2.20

### General-purpose Diode Function Generator

It approximates nonlinear functions of one independent variable,  $y = f(x)$ , by piecewise-linear approximation. In analog computers, it is used as a nonlinearity. It simulates the relationship between voltages, similar to that described by the equation of piecewise-linear approximation, Eq. (2.68):

$$V_{out} = V_0 + kV_{in} + \sum_{i=1}^n b_i (V_{in} - V_{in,i}) \quad (2.75)$$

where  $V_0$  is the initial value of output voltage at  $V_{in} = 0$ ,  $V_{in.i}$  is the input voltage at the start of each approximating segment, and  $k = \tan \alpha_0$ . For  $V_{in} \leq V_{in.i}$ ,

$$b_i = 0$$

and for  $V_{in} > V_{in.i}$ ,

$$b_i = \tan \alpha_i - \tan \alpha_{i-1}$$

The voltages and the machine variables are related as

$$V_{out} = y/M_y, \quad V_0 = y_0/M_y, \quad V_{in} = x/M_x, \quad V_{in.i} = x_i/M_x,$$

The general-purpose function generator shown in Fig. 2.21 incorporates elements mechanizing each term on the right-hand

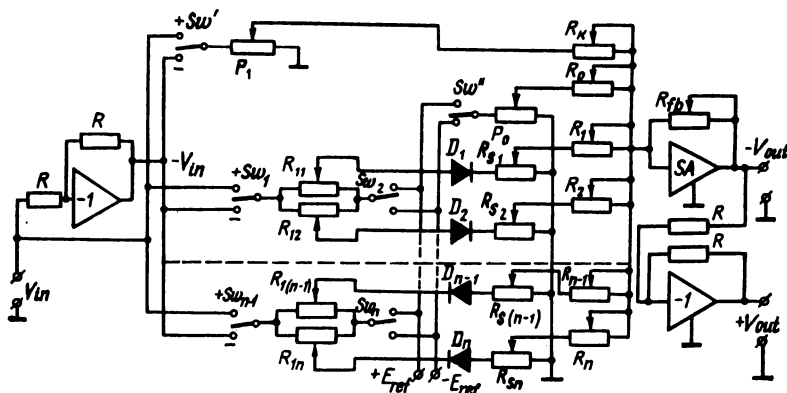


Fig. 2.21

side of Eq. (2.75). The initial voltage,  $V_0$ , is set by the potentiometer  $P_0$ , the term  $kV_{in}$  by another potentiometer,  $P_1$ . The terms of the summation,  $V_i = b_i(V_{in} - V_{in.i})$  are instrumented by biased diode circuits of Types A and B. The switches,  $Sw_i$  ( $i = 1, 2, \dots, n$ ), apply  $V_{in}$  and  $E_{ref}$  in appropriate polarity to the dividers,  $R_{1i}$ .

The breakpoint,  $V_{in.i}$ , is adjusted with the contact arms of the dividers,  $R_{1i}$ , and the slope,  $\alpha$ , of the line segments (that is, the coefficient  $b_i$ ) with the slope potentiometers,  $R_{si}$ . The desired characteristics of the diodes in the various quadrants can be obtained by appropriately setting the switches  $Sw_i$  and connecting the diodes  $D_i$  to the dividers  $R_{1i}$  in various ways. From the slope potentiometers,  $R_{si}$ , the voltages are taken to the inputs of a summing amplifier, SA. The gain for the signal applied to a particular input can be adjusted by varying the value of  $R_i$  with  $R_{fb}$  held constant. By suitably adjusting the value of  $R_{fb}$ , it is

possible to adjust the gain for signals applied to all inputs. This arrangement provides a convenient device for adjustment of the slope and scale factors.

In a very simple and efficient way, a desired function can be piecewise-linearly approximated by a graphical-analytical method. The first step is to plot the function  $V_{out} = f(V_{in})$  on a sheet of profile paper, representing the function  $y = f(x)$  within the specified range of values of  $x$ . The scale factors are found by the equations

$$M_y = y_{max}/V_{out\ max}$$

$$M_x = x_{max}/V_{in\ max}$$

The values of  $M_x$  and  $M_y$  are chosen such that the error of the graphical representation will not exceed 0.03 to 0.05% and be less than the average error of approximation.

The next step is to construct the straight-line segments used for approximation, starting with the initial segment. The values of  $V_0$ ,  $V_{in, i}$  and  $V_{out, i}$  (the coordinates of nodal points), the slope  $\alpha_i$  and the quadrants for each diode circuit are then found from the plot. All of these data are entered in a set-up chart on the basis of which the nonlinearity unit of the analog computer is patched. The approximation procedure is arranged to secure the specified accuracy with a minimum number  $n$  of straight-line segments, to simplify the circuit of the diode function generator. As a rule, the number  $n$  is chosen to be equal to or less than 20.

General-purpose diode function generators may also use diodes returned to virtual ground potential. For example, in the MH-14 analog computer, the general-purpose nonlinearity unit contains 20 such circuits built around silicon diodes.

### *Special-purpose Diode Function Generators*

Simple functions of the form  $y = x^2$ ,  $y = x^3$ ,  $y = \sin x$ , etc., and also high-slope functions, such as typical nonlinearities encountered in automatic control systems (dead bands, backlash, stiction, etc.), can readily be approximated by comparatively simple special-purpose circuits which are combinations of diode networks (with a diode returned to virtual ground potential) and d. c. amplifiers. The diode networks, which may be Type *A* and/or Type *B*, are connected in parallel either at the input or in the feedback path or in both. As examples, Fig. 2.22 shows special-purpose function generators incorporating Type *A* diode networks to implement functions with increasing and decreasing derivatives. In the circuit of Fig. 2.22a, the diodes,  $D_i$  ( $i = 1, 2, \dots, n$ ), remain OFF as long as  $V_{in} = 0$ . With increasing  $V_{in}$ , the diodes consecutively turn ON and bring more and more resistors in the

parallel input circuit. As a result, the gain of the opamp rises with each approximation step, thereby simulating a function with an increasing derivative. In the circuit of Fig. 2.22b, the diode networks are placed in the feedback path of the amplifier. As  $V_{in}$  rises, the diodes turn ON consecutively, and the overall resistance of the feedback circuit and, as a consequence, the gain of the amplifier goes down.

Diode function generators coupled with d.c. amplifiers have found use in automatic control systems to implement various characteristics. Consider some of such circuits.

On-off (relay) response and Coulomb friction (voltage limiting) can conveniently be mechanized with the circuit of

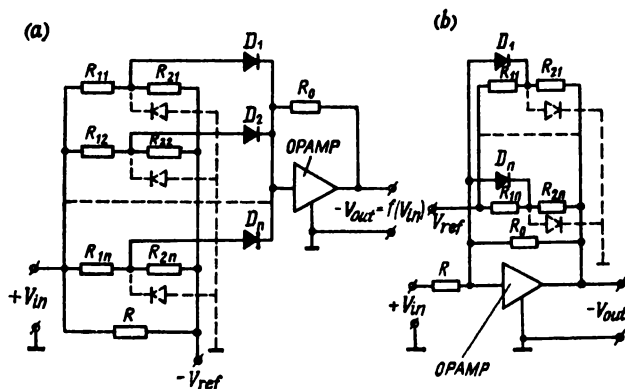


Fig. 2.22

Fig. 2.23a. At  $V_{in} = 0$ , the diodes  $D1$  and  $D2$  are held at cut-off with voltages  $+E$  and  $-E$ . As a result,  $V_{out} = 0$  and  $V_B = 0$ . At small positive values of  $V_{in}$ , the diodes remain in the OFF state, and the circuit is a conventional amplifier for which

$$V_{out} = -(R_0/R_1) V_{in}$$

that is, the output voltage is a linear function of the input voltage (Fig. 2.23b). At  $V_{out} = -E$ , the diode  $D1$  turns on, and the summing junction  $\Sigma$  accepts voltages from the two inputs,  $V_{in}$  and  $E_{ref}$ , via the resistors  $R1$  and  $r1$ , and the resistance of the feedback path (for  $R_{D1} \approx 0$  and  $R_{D2} \approx 0$ ) becomes

$$R_{fb} = R_0 r_0 / (R_0 + r_0)$$

As a result, the gain is drastically reduced, and the output voltage is defined as

$$V_{out} = -R_{fb} \left( \frac{1}{R_1} V_{in} + \frac{1}{r_1} E_{ref} \right) = -\frac{(r_1/R_1) V_{in} + E_{ref}}{r_1 (R_0 + r_0) / R_0 r_0} \quad (2.76)$$

By setting  $r_0 \ll R_0$  and  $r_1 \ll R_1$ , we get

$$V_{out} \approx |(r_0/r) E_{ref}| = \text{constant}$$

that is,  $V_{out}$  remains practically unchanged with a further increase in  $V_{in}$ .

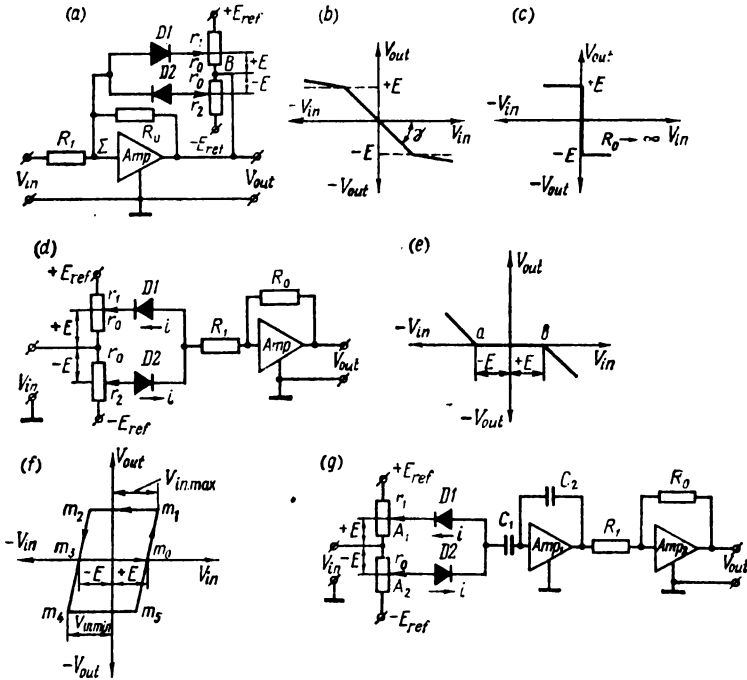


Fig. 2.23

$V_{out}$  is limited at  $V_{in} < 0$  in a similar way. Setting  $R_0 = \infty$ , for  $V_{in} = 0$  we have

$$V_{out} = |(r_0/r) E_{ref}|$$

that is, even a small signal at the input will cause the output voltage to rise to its maximum value,  $V_{out} = |\pm E|$ , and to remain unchanged afterwards despite any increase in  $V_{in}$  (Fig. 2.23c). This is a simulation of on-off (relay) response and Coulomb friction.

The behaviour of an element having a dead zone can be simulated by the circuit of Fig. 2.23d. At  $0 \leq V_{in} < |E|$ , the diodes

$D1$  and  $D2$  are OFF, and  $V_{out} = 0$ . At  $V_{in} \geq |-E|$ , the diode  $D2$  turns on, and  $V_{out}$  varies linearly (Fig. 2.23e). Similar events occur when  $V_{in}$  is less than zero. The interval  $ab$  on the characteristic curve is called a dead zone or band.

The behaviour of an element with backlash or hysteresis (Fig. 2.23f) can be simulated by the circuit of Fig. 2.23 g. At  $0 < V_{in} < |-E|$ , both diodes are OFF, and  $V_{out} = 0$ . When  $V_{in}$  becomes equal to  $|-E|$ , the diode  $D2$  turns on (at point  $m_0$ ), and now the output voltage behaves as a linear function of the input voltage ( $m_0m_1$ ):

$$V_{out} = -(C1/C2) V_{in}$$

A decrease in  $V_{in}$  will cause the voltage  $V_{C1}$  across capacitor  $C1$  to turn off the diode  $D2$ . As a result, the charge accumulated by the capacitors  $C1$  and  $C2$  will remain unchanged and, as a consequence, the output voltage will remain constant (interval  $m_1m_2$ ) until the diode  $D1$  is turned on by the negative input voltage

$$|V_{in}| = |V_{in\max} - 2E|$$

(point  $m_2$ ), when its anode and cathode come by the same potential.

The further change in  $V_{in}$  will cause the capacitors  $C1$  and  $C2$  to re-charge. Starting at the instant when  $V_{in} = -E$  (point  $m_3$ ), the events will be repeated all over again, but in the reverse sequence ( $m_3m_4m_5m_0$ ). The other amplifier,  $Amp2$ , is used for slope adjustment.

The above circuit may well be used to simulate magnetic hysteresis approximately.

Instead of conventional valves and crystal diodes, function generators may use silicon Zener diodes with a characteristic shown in Fig. 2.24a, which need no reference voltage for their operation. A major distinction of Zener diodes from ordinary crystal diodes is that their resistance is very high (anywhere between 10 and 20 megohms) at  $-V_{BD} < V_Z < 0$  (where  $V_{BD}$  is the breakdown voltage), and is extremely low at  $V_{in} < -V_{BD}$  (of the order of 10 to 30 ohms). In the latter case the characteristic shows what is known as the Zener knee which marks the onset of an electric (reversible) breakdown and a sharp break from non-conduction to conduction. The forward part of the characteristic (relating the forward voltage and forward current) has a similar shape. To minimize the effect of the forward characteristic, it is usual to connect a Zener diode back to back with a conventional diode (Fig. 2.24b).

Figure 2.24c shows the circuit of a three-terminal diode function generator using a Zener diode. As already explained, the Zener



diode shows a sharp break to conduction when the input voltage becomes

$$V_{in}^{on} = \frac{R_1 + R_2}{R_2} V_{BD}$$

where  $V_{BD}$  is the breakdown voltage as before. As is seen, the position of the Zener knee can readily be adjusted by varying the values of  $R_1$  and  $R_2$ . In the conducting state, the function generator shows a linear response,  $i = f(V_{in})$ .

Function generators using Zener diodes are arranged in about the same circuit configurations as function generators using conventional diodes, but are more economical.

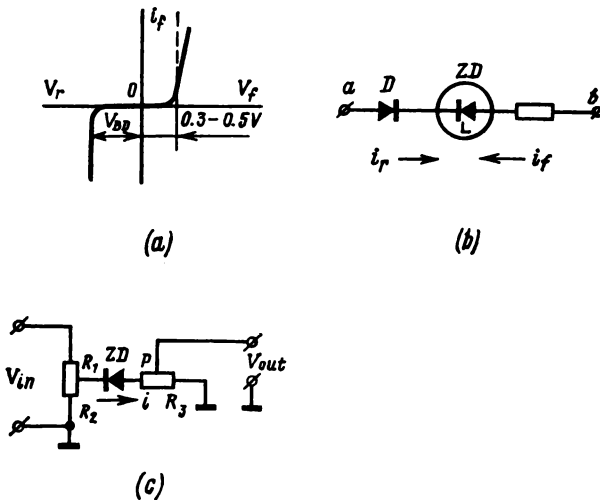


Fig. 2.24

**2.5.5. Time function generators using stepping switches.** In a very simple manner, a stepping switch can be set up in a circuit which can approximate time-varying functions, for example, the variable coefficients,  $\alpha(t)$ , of equations by 'piecewise-constant (staircase) approximation.

The stepping-switch function generator of Fig. 2.25a consists of a two-section voltage divider,  $VD$  (with a hundred resistors of value  $r$  in each section), and a stepping switch,  $SS$ , with a hundred contact bars. The voltage divider reproduces the ordinates of the function, and the stepping switch provides a time base. The finger of the stepping switch moves across the bank of contact bars at a certain fixed rate maintained by a clock generator and determining the duration of time intervals.

The contact bars of the stepping switch are connected to the respective taps on the voltage divider by means of patch cords inserted into the appropriate voltage-divider and stepping-switch sockets on a patch board. As a rule, the  $x$ -axis is divided into 100 increments, and the  $y$ -axis into 200 ( $\pm 100$ ) increments (Fig. 2.25b). The maximum value of the function (coefficient)

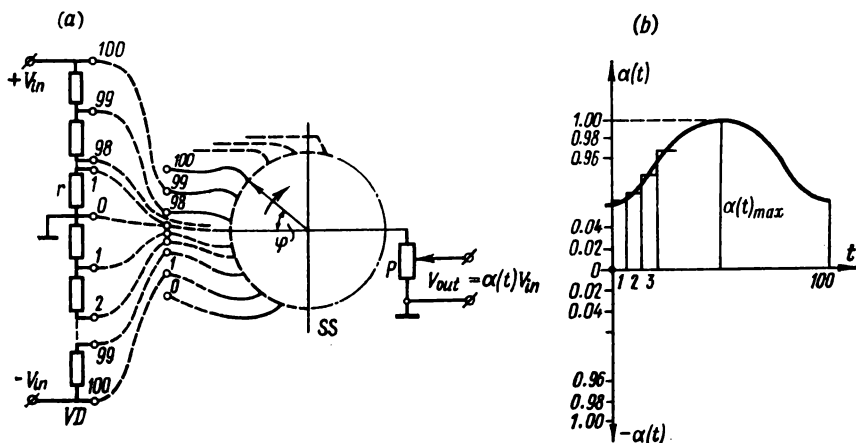


Fig. 2.25

should not exceed unity. The voltages,  $V$ , simulating the ordinates of the function  $\alpha(t) = f(t)$ , are derived from the condition

$$\alpha(t)/\alpha(t)_{\max} = V/V_{\max} \quad (2.77)$$

whence

$$V = \alpha(t) V_{\max} = 100\alpha(t)$$

Because

$$V_{\max} = 100V$$

then

$$\alpha(t)_{\max} = 1$$

The accuracy with which functions varying in slope can be approximated is enhanced by using time intervals which decrease with increasing slope.

**2.5.6. Shaped-potentiometer function generator.** A given non-linear function can be approximated by a continuously-wound potentiometer in several ways. One of the most commonly used techniques is that based on shaped-card (or simply, shaped) or tapered potentiometers.

As its name implies, a shaped-card potentiometer (Fig. 2.26) has its continuous resistance element wound on a card shaped so that the output voltage,  $V_{out}$ , is proportional to a specified mathe-

mathematical function,  $y = f(x)$ . The independent variable,  $x$ , is set by the position of the slider.

The design of a shaped-card potentiometer reduces mainly to finding an equation to describe the shape of the card,  $h = F(x)$ . Let the slider move from a position,  $x$ , by a distance  $\Delta x = d$  where  $d$  is the wire diameter. Then the resistance will change by

$$\Delta R = (dr_x/dx) \Delta x = (dr_x/dx) d$$

Defining the change of resistance as

$$r_x = R_t f(x)$$

where  $R_t$  is the total resistance of the potentiometer, we get

$$\Delta R = R_t d [df(x)/dx]$$

The turn length may be found from any one of two relations:

$$l_1 = \Delta R / \rho_0 = (R_t d / \rho_0) [df(x)/dx] \quad (2.78)$$

or, leaving out the wire diameter,

$$l_1 \approx 2(h + b) \quad (2.79)$$

where  $\rho_0$  is the resistance per unit length of wire, and  $b$  is the thickness of the card.

From Eqs. (2.78) and (2.79), we have

$$h = R_t d / 2\rho_0 [df(x)/dx] - b = F(x) \quad (2.80)$$

The total resistance,  $R_t$ , is found to satisfy the requirement for the maximum power dissipation,  $P_{lim}$

$$R_t \geq V_0 / P_{lim}$$

After the material for the resistance element has been selected (which may be constantan, manganin or Ni-Cr wire), the wire diameter  $d$  is determined to satisfy the requirement for the maximum safe current density,  $\Delta I_{lim}$  (which should be 2 or 3 A/mm<sup>2</sup>):

$$d \geq 2 \sqrt{V_0 / \pi R_t \Delta I_{lim}}$$

As a rule,  $d = 0.05$  to  $0.5$  mm. The values of  $\rho_0$  are taken from tables, and the total length,  $L$ , of the resistance element is decided by construction considerations. The thickness of the card is  $b = 0.3$  to  $2$  mm, its minimum height  $h_{min} = 2$  or  $3$  mm, and its maximum height  $h_{max} = 50$  to  $60$  mm.

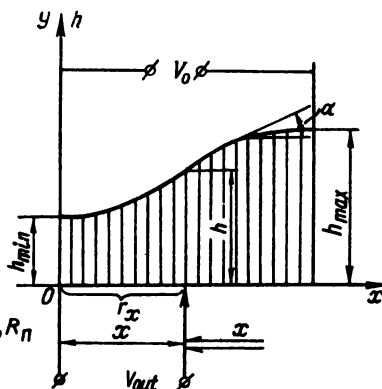


Fig. 2.26

A disadvantage of shaped-card potentiometers is that the angle of slope,  $\alpha$ , of the card is limited ( $20^\circ$  to  $30^\circ$ ) because of which they cannot approximate fast-varying functions. Furthermore, an increase in  $\alpha$  (that is, in the rate of resistance change) complicates the winding operation and leads to some other difficulties.

Where a function shows an irregular behaviour (for example, it may tend to zero or infinity for certain values of the independent variable) so that the card has to be given an unusual shape, special measures need to be taken to maintain normal operation of the function generator.

**2.5.7. Cathode-ray tube function generators.** Of all CRT function generators, the most commonly used type is the photoformer.

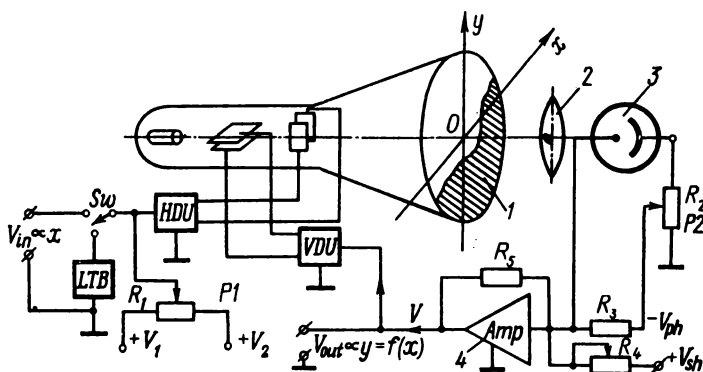


Fig. 2.27

A simplified circuit of a practical photoformer is shown in Fig. 2.27.

Referring to Fig. 2.27, at 1 is a mask prepared as a plot of the function  $y = f(x)$  desired, and the area below the curve is made opaque. The vertical position of the electron beam is made to follow the edge of the mask by a feedback servo system which is a combination of an optical focusing system, 2, a photo-cell, 3, an amplifier, 4, a vertical deflection unit, VDU, and the vertical deflection plates. The positive bias voltage,  $V_{bias}$ , at the input to the summing amplifier, SA, tends to move the electron beam upwards, while the output voltage of the photo-cell,  $V_{pc}$ , tends to move it downwards. If the beam is below the edge of the mask, then  $V_{pc} = 0$ , and  $V_{bias}$  is free to move it upwards. As soon as the beam spot appears above the edge, the photo-cell generates a voltage,  $V_{pc} \neq 0$ , with the result that a voltage

$$V = k(V_{bias} - V_{pc})$$

appears across the amplifier output (where  $k$  is the gain of the summing amplifier).

As the beam spot keeps moving upwards, its area expands, and  $V_{pc}$  rises until it becomes equal in its effect to  $V_{bias}$ . At that instant the beam will be positioned on the edge of the mask, thereby giving the value of  $y$  corresponding to a given value of  $x$ . Since  $y$  is proportional to  $V$ , its value is picked off as voltage.

The horizontal displacement of the beam is produced by the input voltage adjusted to be proportional to  $x$  and applied to the horizontal deflection plates via the time-base unit, *TBU*.

Due to the joint action of the two voltages, the beam is thus constrained to follow the curve on the mask, thereby reproducing the function desired.

If use is made of a linear time-base, *LTB*, the result will be  $V_{out} = f(t)$ . The horizontal shift of the coordinate origin is obtained with a potentiometer, *P1*, and the vertical shift, by adjustment of  $V_{bias}$ .

As a matter of convenience, a single mask may be replaced with a folding screen made up of narrow rectangular strips, each being a plot of a different function. As a further refinement which avoids the limit set to the range of values of  $x$  by the *CRT* face, the electron beam may be held stationary, and a perforated transparent film on which the desired function is plotted may be moved past the beam spot. As the film is moved along the  $Ox$ -axis, a special mechanism causes the beam to move up and down in a vertical slit (along the  $Oy$ -axis) or to stay on the opaque curve of the plot, thereby reproducing the function.

*CRT* function generators are versatile, have a fast response, and show satisfactory accuracy (down to 1%).

**2.5.8. Special-purpose function generators.** A variety of non-linear functions can with sufficient approximation be generated by special-purpose circuits of fairly simple configuration based on linear potentiometers, nonlinear valves, thyristors, etc. As a rule, the desired function is approximated by the interpolation method. The circuit parameters of the function generator are chosen such that the functional relationship they generate has common nodal points with the desired function, while the difference (error) at other points is within predetermined limits.

An example of a special-purpose function generator is the circuit shown in Fig. 2.28a, using a loaded linear potentiometer. For this circuit we may write

$$U_{out}/V_{in} = \alpha\beta/(\alpha + \beta - \beta^2)$$

where  $\alpha = R_L/R$  and  $\beta = r/R$ . With the value of  $\alpha$  varied between 0 and 1, a series of curves,  $V_{out}/V_{in} = f(\beta)$ , of different slope (shown by the solid lines in Fig. 2.28b) are generated.



SiC varistors  $\beta = 1$  to 3.5, with  $V$  ranging from zero to 100 V). According to values of  $\beta$ , varistors may show quadratic, cubic and other transfer characteristics over a relatively narrow range. To extend the range of values for  $V$  over which a desired transfer characteristic can be obtained, suitably matched linear potentiometers may be connected in parallel or series with varistors.

A circuit generating the functions  $V_{out} = kV_{in}^2 = kV_{in}^3 = \dots = kV_{in}^n$  (for  $\beta = 2, 3, \dots, n$ ) is shown in Fig. 2.28c. Resistors  $R_1$  and  $R_2$  reduce power dissipation across the varistor and stabilize operation of the circuit. A thermistor may be placed in series with  $R_2$  or in the feedback path in order to minimize temperature-change effects on the varistor.

## 2.6. Function Multipliers-Dividers

**2.6.1. General.** A function multiplier and a function divider produce an output which is a function of (a) the product of two variables and (b) the quotient of two variables, respectively, that is,  $z = f(x, y)$ , or expressing the variables in terms of voltages:

$$V_{out} = kV_x V_y \quad (2.81)$$

$$V_{out} = k_1 V_x / V_y \quad (2.82)$$

where  $V_x$  and  $V_y$  are the input voltages proportional to  $x$  and  $y$ .

All function multipliers-dividers may be classed into two groups: (a) explicit, that is, those providing multiplication or division directly and (b) implicit, that is, those in which the operations of multiplication and division are replaced with other mathematical operations, such as

quarter-square:

$$xy = \frac{1}{4} [(x + y)^2 - (x - y)^2] \quad (2.83)$$

logarithmic:

$$\ln xy = \ln x + \ln y \quad (2.84)$$

trigonometric:

$$xy = \frac{1}{2} [\cos(a - b) - \cos(a + b)] \quad (2.85)$$

where  $x = \sin a$  and  $y = \sin b$  for  $|x, y| < 1$

integrating:

$$xy = \int x dy + \int y dx, \text{ etc.} \quad (2.86)$$

Practical circuits are usually based on Eq. (2.83). As a rule, division is performed by a multiplier through implicit computation (inverse multiplication). This can be done in any one of two ways.

(1) A function generator,  $FG$ , generating a quantity which is

the reciprocal of one of the terms is placed at one of the inputs to a function multiplier, *FM* (Fig. 2.29a). The output voltage of the entire circuit will then be

$$V_{out} = k_2 k_1 (1/V_x) V_y = c V_y / V_x$$

where *c* is a proportionality factor.

(2) A multiplier is placed in the feedback path of a high-gain d.c. amplifier (Fig. 2.29b). Then it may be assumed that  $i_s = 0$ ,

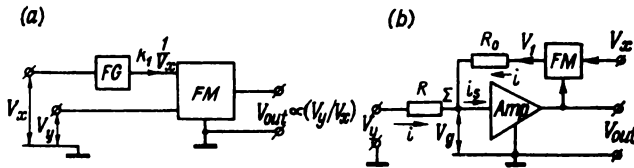


Fig. 2.29

and we may write

$$(V_y - V_g)/R + (V_1 - V_g)/R_0 = 0 \quad (2.87)$$

Substituting

$$V_g = -V_{out}/k$$

$$V_1 = a V_x V_{out}$$

in Eq. (2.87), we get after simple manipulations

$$V_y R_0 + V_{out} \left[ \frac{1}{k} (R_0 + R) + R a V_x \right] = 0$$

In the limit of high gain values, the first term in the brackets may be neglected, in which case

$$V_{out} = - (R_0 / R a) (V_y / V_x) = - c V_y / V_x \quad (2.88)$$

**2.6.2. Potentiometer multipliers-dividers.** Two or more alternating quantities can be multiplied together by a circuit composed of several stages of grounded-centre-tap linear potentiometers. One such circuit to multiply together two variables, *x* and *y*, is shown in Fig. 2.30a. For the voltage *V<sub>x</sub>* at the output of the potentiometer *P<sub>1</sub>* to be linear, it is important that  $R_2 \gg R_1$ . Then

$$V_x = V_0 (r_x / R_1) = V_0 (\phi_1 / \phi_{1\max})$$

where  $r_x$  = the setting of the potentiometer *P<sub>1</sub>* proportional to *x*

$\phi_1$  and  $\phi_{1\max}$  = the current and maximum shaft positions of the potentiometer (if *x* and *y* are set as shaft positions)

The output voltage of the circuit will then be

$$V_{out} = V_x (r_y / R_2) = (V_0 / R_1 R_2) r_x r_y$$



or

$$V_{out} = V_x (\phi_2 / \phi_{2max}) = V_0 \phi_1 \phi_2 / \phi_{1max} \phi_{2max} \quad (2.89)$$

Thus, when  $\phi_1$  and  $\phi_2$  are set to be proportional to  $x$  and  $y$ ,  $V_{out}$  will be proportional to the product

$$z = cxy$$

In the above arrangement, the value of  $V_{out}$  is affected by variations in the supply voltage,  $V_0$ . Besides, when several va-

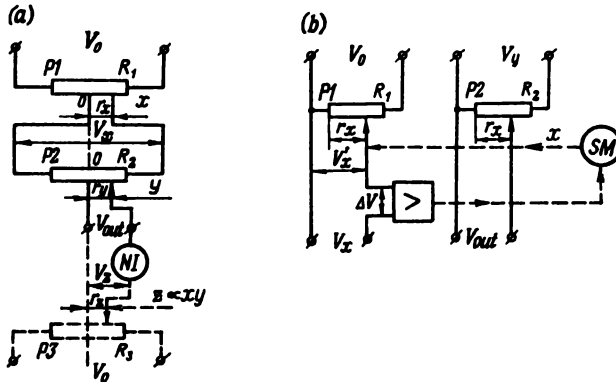


Fig. 2.30

riables are multiplied together, the resistance of the potentiometer in the last stage may be very high because

$$R_n \gg R_{n-1} \gg \dots \gg R_2 \gg R_1$$

where  $n$  is the number of stages. These factors give rise to an error. Variations in  $V_0$  may be counteracted by applying  $V_{out}$  to a null-indicator which also accepts  $V_z$  picked off the slider of a potentiometer,  $P_3$  (shown by the dotted lines in Fig. 2.30a). As long as the slider is positioned so that  $V_{out} = V_z$ , its position will be proportional to the product  $xy$ .

The above circuit can readily be converted to a servo multiplier-divider by applying the difference (or error) voltage,  $\Delta V = V_{out} - V_z$ , to a servo amplifier which controls a servo motor coupled to the slider of the potentiometer  $P_3$ . Apart from multiplication, the circuit can be used for division, squaring, and extraction of the square root (for  $x = y$ ), with the inputs and outputs appropriately interchanged.

A servo potentiometer multiplier-divider is shown in Fig. 2.30b where the variables to be multiplied together are set as voltages,  $V_x$  and  $V_y$ . After the variables are set, the servo system positions

the sliders of the potentiometers  $P1$  and  $P2$  to  $r_x$ . When all transients die out,

$$\Delta V = V_x - V'_x = 0 \quad (2.90)$$

Since

$$V'_x = (r_x/R_1) V_0$$

then, on finding  $r_x$  and substituting it in the expression for  $V_{out}$ , we get

$$V_{out} = (r_x/R_2) V_y = (R_1/R_2 V_0) V_x V_y = k V_x V_y \quad (2.91)$$

**2.6.3. Time-division multipliers-dividers.** The basic principle of time-division multiplication consists in determining the average

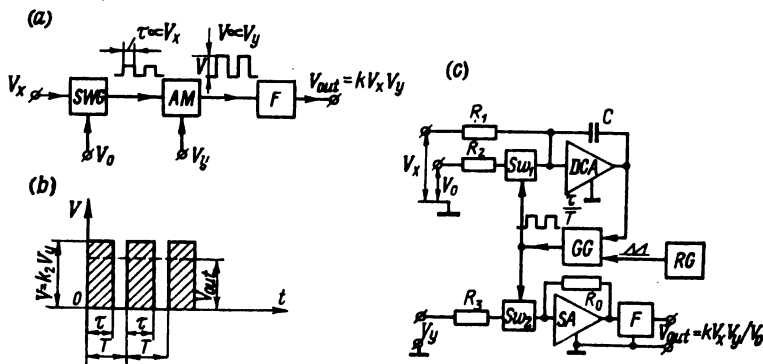


Fig. 2.31

amplitude of a voltage consisting of a train of rectangular pulses and proportional to the product of two input variables, one being made proportional to the ratio of pulse duration to pulse period and the other to the pulse height.

A simplified circuit of a time-division multiplier is shown in Fig. 2.31a. A square-wave generator, SWG, converts the input voltage,  $V_x$ , into a wave of rectangular pulses with a duration,  $\tau$ , proportional to  $V_x$ , and with a period  $T$  held constant. Then the ratio of pulse duration to pulse repetition period (pulse duty factor) will be

$$k_d = \tau/T = k_1 V_x$$

An amplitude modulator, AM, which accepts pulses from the SWG and also the voltage  $V_y$  proportional to the other variable, modulates the pulses in height (Fig. 2.31b) so that

$$V = k_2 V_y$$

The average value of voltage across the output of the smoothing filter,  $F$ , is represented by the area under a pulse over the

period  $T$ :

$$V_{out.av} = V\tau/T = k_1 k_2 V_x V_y = k V_x V_y$$

The circuit of a practical time-division electronic multiplier-divider with feedback is shown in Fig. 2.31c. It incorporates a d. c. amplifier, *DCA*, a gate generator, *GG*, a ramp generator, *RG*, and an electronic switch *Sw1*, which form between them a channel which converts  $V_x$  and  $V_0$  to the gain of a scaling amplifier, *SA*.  $V_x$  and  $V_0$  are compared, and their difference,  $\Delta V = V_0 - V_x \times (V_0 \geq V_x)$  is boosted by the d. c. amplifier and applied to the gate which also accepts triangular pulses from the ramp generator. The gate generator is essentially a bistable multivibrator. Its output is a train of rectangular gating pulses whose duty factor (pulse duration to pulse period ratio) is given by

$$k_d = \tau/T = k_1 V_x / V_0$$

These pulses cause the electronic switches *Sw1* and *Sw2* to close and open periodically, thereby controlling the average conductance of the input circuit in the scaling amplifier, *SA*, according to  $\tau/T$ . When the input to *SA* is  $V_y$ , then (after smoothing by the filter *F*) its output voltage will be

$$V_{out} = k_2 (\tau/T) V_y = k (V_x V_y / V_0) \quad (2.92)$$

For  $V_0$  held constant, the device operates as a multiplier. If, on the other hand,  $V_0 \neq \text{constant}$ , both multiplication and division can be provided. The sign of  $V_{out}$  is matched by a suitable circuit as the polarity of  $V_x$  and  $V_y$  is reversed.

The accuracy of a time-division electronic multiplier-divider of the type shown in Fig. 2.31c (which is, incidentally, used in the Soviet-made MH-14 analog computer) is of the order of 0.15% at a gating-pulse repetition frequency of 5 kHz and 0.3% at a gating-pulse frequency of 10 kHz. Among the demerits of time-division multipliers-dividers are circuit complexity, a limited bandwidth and, as a consequence, a considerable time lag due to the need to provide an output filter.

**2.6.4. AM and FM multipliers.** The bandwidth can markedly be extended in comparison with that of time-division multipliers through the use of amplitude and frequency modulation. Modulation is applied twice: one of the variables modulates the carrier in, say, amplitude, and the other in frequency. In fact, the same or different forms of modulation may be used, giving rise to *AM-AM*, *AM-FM* or *FM-FM* multipliers.

In an *AM-AM* multiplier (Fig. 2.32a), a carrier frequency, at a constant amplitude,  $V_c$ , generated by a harmonic oscillator, *HO*, is fed to an amplitude modulator, *AM1*, which also accepts the

voltage,  $V_x$ , representing one of the variables,  $x$ . The modulated voltage is then

$$V_1 = k_1 V_x \sin \omega t$$

Together with  $V_y$  representing the other variable,  $V_1$  is then applied to a second amplitude modulator,  $AM2$ , whose output voltage is

$$V_2 = k_2 V_1 V_y = k_1 k_2 V_x V_y \sin \omega t$$

After it is rectified by a detector,  $D$ , and smoothed by a filter,  $F$ , the average d. c. voltage at the output of the multiplier is

$$V_{out} = k V_x V_y$$

An  $AM$ - $FM$  multiplier in block diagram form is shown in Fig. 2.32b. One of the variables represented by  $V_x$  is used to

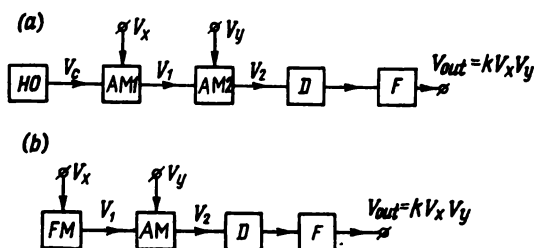


Fig. 2.32

modulate the carrier frequency,  $\omega$ , of a sinusoidal voltage,  $V_1$ , through a frequency modulator,  $FM$ . The other variable represented by  $V_y$  modulates the carrier amplitude through an amplitude modulator,  $AM$ . The amplitude modulator may be an r. f. amplifier whose gain is proportional to  $V_y$ . The average output voltage will be

$$V_{out av} = k_1 A \omega = k V_x V_y$$

where

$$V_x \propto \omega$$

and

$$V_y \propto A$$

**2.6.5. Hall-effect multiplier.** This type of multiplier uses a Hall generator, a thin wafer of a semiconductor material (such as indium arsenate or indium antimonate) arranged to be at right angles to a magnetic field of induction  $B$  (Fig. 2.33a). If a current,  $i$ , be passed through the wafer, a Hall voltage,  $V_H$ , will be set up between points  $m$  and  $n$ , defined as

$$V_H = k_H (iB/h) \quad (2.93)$$

where  $k_H$  is the Hall constant ( $\text{cm}^3 \text{ A}^{-1} \text{ s}^{-1}$ ), and  $h$  is the wafer thickness (cm).

If  $i$  is set to be proportional to  $V_y$  and  $B$  to  $V_x$ , the Hall generator can readily be used to multiply together the two voltages

$$V_H = c V_x V_y$$

where  $c$  is a constant.

The circuit of a practical feedback Hall-effect multiplier is shown in Fig. 2.33b. It uses two Hall generators,  $HG1$  and  $HG2$ . The second Hall generator is placed in the air gap of the same magnetic circuit and is energized with a reference voltage,  $V_0$ . D.c. amplifiers,  $DCA1$  and  $DCA2$ , convert  $V_x$  and  $V_y$  representing the variables to be multiplied together, into currents,  $i_x$  and  $i_y$ ,

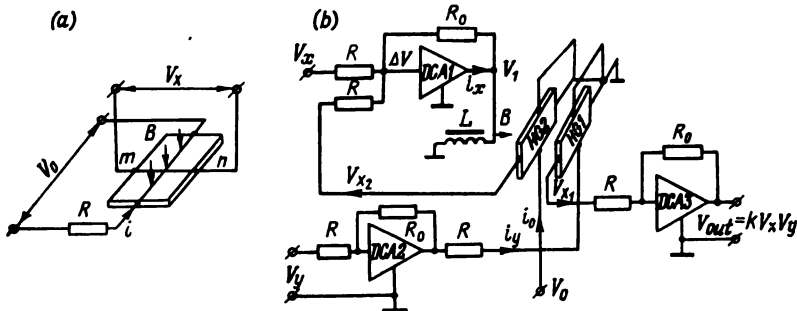


Fig. 2.33

which should be sufficiently high (tens or even hundreds of milliamperes) for the first Hall generator to deliver an output voltage of a few tens of millivolts. The current  $i_x$  energizes the coil  $L$  of the magnetic circuit, and the current  $i_y$  flows through the first Hall generator,  $HG1$ . The Hall voltage,  $V_{H1}$ , generated by the second Hall generator,  $HG2$ , is applied to the first d.c. amplifier to be compared with  $V_x$ . As a result, the amplifier output voltage is

$$V_1 = -k(V_x - V_{H1})$$

which is impressed on the magnetic-circuit coil to set up a field of an induction defined as

$$B = c_1 V_1 = c_1 k (V_x - k_{H1} i_0 B / h) \quad (2.94)$$

where  $c_1 = \text{constant of proportionality}$

$$V_{H1} = k_{H1} i_0 B / h \text{ according to Eq. (2.93)}$$

On finding  $B$  from Eq. (2.94) and noting that  $h \ll c_1 k k_{H1} i_0$ , we get

$$B = \frac{h}{k_{H1} i_0} V_x$$

The Hall voltage picked off  $HG1$  may be defined as

$$V_{H1} = k_{H1} \frac{i_y B}{h} = \frac{k_{H1}}{k_{H1} i_0} i_y V_x = \frac{c_2}{i_0} V_x V_y$$

where  $i_y = c_2 V_y$  and  $k_{H1} = k_{H2}$ .

The output voltage of the multiplier is

$$V_{out} = -(R_0/R) V_{H1} = -(c_2 R_0/iR) V_x V_y = -c V_x V_y \quad (2.95)$$

Although Hall generators have a broad bandwidth (of the order of  $10^{12}$  radians/s), the presence of amplifiers and an electromagnet in the circuit imposes stringent limitations. The accuracy of the circuit is up to 1%.

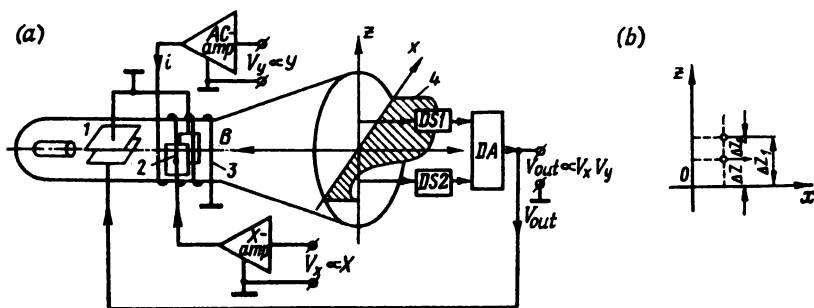


Fig. 2.34

**2.6.6. Crossed-fields electron-beam multiplier.** This device operates on the crossed fields in a cathode-ray tube, and is illustrated in Fig. 2.34a. Voltages proportional to  $x$  and  $y$ , that is,  $V_x$  and  $V_y$ , are applied to the horizontal-deflection amplifier,  $X\text{-amp}$ , and the axial-coil amplifier,  $AC\text{-amp}$ , respectively whence the amplified  $V_x$  is applied to the horizontal deflection plates, 2, and the amplified  $V_y$  to the axial coil, 3, around the neck of the cathode-ray tube. At  $V_x = 0$ , the concentric magnetic field due to  $V_y$  holds the beam aligned with the axis of the CRT ( $\Delta x = 0$  and  $\Delta y = 0$ ). At  $V_y = 0$  (the concentric magnetic field is  $B = 0$ ),  $V_x$  gives the electron beam a component of velocity directed along the  $Ox$  axis. In the presence of both  $V_x$  and  $V_y$ , the cross product of horizontal velocity and concentric magnetic field deflects the beam along both the  $Ox$  and  $Oz$  axes the deflection along the  $Oz$  axis (vertically) being proportional to the product  $xy$ . This vertical deflection is detected by error detectors and converted to a voltage by a feedback circuit to keep the beam horizontal as follows.

In deflecting vertically, the beam strikes either an "up" deflection sensor,  $DS1$ , or a "down" deflection sensor,  $DS2$ , separated

by an opaque screen, 4. The sensor signal is then applied to a differential amplifier, *Damp*, which feeds its voltage to the vertical deflection plates, 1, so as to cause the beam to deflect in the opposite direction until a balance struck between the deflection forces.

The total vertical deflection of the beam (Fig. 2.34*b*) is

$$\Delta z = \Delta z_1 - \Delta z_2 = c_1 V_x V_y - c_2 V_{out} \quad (2.96)$$

where  $\Delta z_1$  = vertical beam deflection with  $V_x$  and  $V_y$  applied

$c_1, c_2$  = constants of proportionality

$\Delta z_2$  = vertical beam deflection due to  $V_{out}$

Since

$$V_{out} = c_3 k \Delta z \quad (2.97)$$

where  $k$  is the gain of the differential amplifier and  $c_3$  is a constant of proportionality, then, substituting (2.96) in (2.97) gives:

$$V_{out} = \frac{c_1 c_3 k V_x V_y}{1 + c_2 c_3 k} \approx (c_1/c_2) V_x V_y \quad (2.98)$$

assuming that  $c_2 c_3 k \gg 1$ . With  $V_{out}$  applied to plates 2,  $V_x$  to plates 1, and  $V_y$  to the concentric magnetic field coil, 3, the circuit will carry out the operation of division

$$V_{out} = c V_x / V_y$$

Since it uses no lag elements, the circuit has a high speed of response and a satisfactory frequency response (up to 20 kHz for  $V_x$  and up to 3 kHz for  $V_y$ ). The multiplication error may be reduced down to 0.5 or 1%. A disadvantage of this type of multiplier is an elaborate circuitry.

**2.6.7. Quarter-square multipliers.** This type of multiplier implements a relation of the form given by Eq. (2.83). The main differences in mechanizing this equation are due to differences in the circuits used to instrument the terms  $(x + y)^2$  and  $(x - y)^2$ , that is, squaring circuits. In schematic form, one of the simplest and most commonly used quarter-square multipliers is shown in Fig. 2.35*a*. It mechanizes the relationship

$$V_{out} = c \frac{1}{4} [(V_x + V_y)^2 - (V_x - V_y)^2] = c V_x V_y \quad (2.99)$$

where  $V_x$  is proportional to  $x$ ,  $V_y$  to  $y$ ,  $V_{out}$  to  $z$ , and  $c$  is a constant of proportionality

The applied voltages are added together algebraically by four simple two-input current summing networks (see Fig. 2.11*a*).

The first two summing networks form  $\pm \frac{V_x + V_y}{2}$ , and the remaining two form  $\pm \frac{V_x - V_y}{2}$ . For any combination of values and

signs of  $V_x$  and  $V_y$ , one of the first two summing networks will deliver a positive half-sum, and one of the second pair will put out a negative half-sum. The circuit of Fig. 2.35a applies to a case where  $V_x > V_y$ , and the diodes  $D1$  and  $D3$  are conducting. The positive half-sum is applied via  $D1$  (or  $D2$ ) to a squarer,  $SQI$ ,

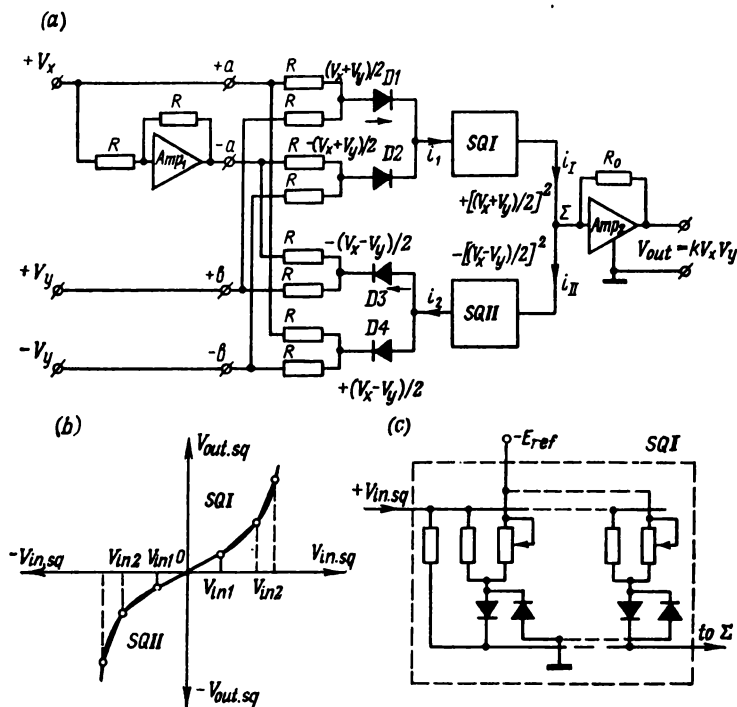


Fig. 2.35

and the negative half-difference to another squarer,  $SQII$ . The two squarers are connected to the summing junction of an output amplifier. As a result, the output amplifier (a summer) delivers  $V_{out}$  defined by Eq. (2.99).

The squarers are usually diode function generators approximating a parabola located in the 1st and 3rd quadrants (Fig. 2.35b).

The circuit of squarer I using crystal diodes clamped to virtual ground potential is shown in Fig. 2.35c. As the squarer input,  $V_{in.sq}$ , proportional to  $(V_x + V_y)/2$  rises to a certain value,  $V_{in.t}$ , all the  $n$  diodes of the squarer are rendered conducting. Each time a diode is turned on, the total resistance of the squarer goes down, and the slope of the curve  $i_1 = f(V_x + V_y)$  increases. As



a result, a current begins to flow, given by

$$i_1 = \frac{c}{4} (V_x + V_y)^2 = c_1 V_{in. sq}^2$$

Similarly, the current,  $i_2$ , due to the second squarer, SQII, will be

$$i_2 = \frac{c}{4} (V_x - V_y)^2$$

The frequency response of the quarter-square multiplier discussed above is mainly decided by the capabilities of operational amplifiers (from a few hundred to a few thousand hertz). The per cent error may be reduced to 0.6 to 0.8%. The approximation error of the squarers is 0.1 to 0.15%.

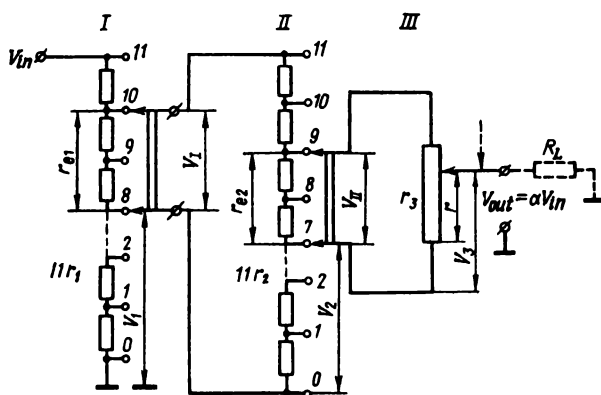


Fig. 2.36

Replacement of crystal diodes with silicon-carbide varistors having a square-law volt-ampere characteristic produces more compact and economical multipliers.

**2.6.8. Coefficient-setting potentiometers.** In analog computers, an a.c. input voltage is most often multiplied by a constant positive coefficient,  $\alpha \leq 1$ , on a resistive voltage divider, or potentiometer (Fig. 2.36). Stages I and II of the divider have each eleven identical resistors. Stage III is a linear potentiometer. The resistances are chosen to satisfy the conditions

$$r_3 = 2r_2$$

$$r_2 = 0.2r_1$$

The equivalent resistances between the arms of stages I and II respectively are

$$r_{eq2} = 2r_2 r_3 / (2r_2 + r_3) = 2r_2 2r_2 / (2r_2 + 2r_2) = r_2$$

$$r_{eq1} = 2r_1 10r_2 / (2r_1 + 10r_2) = 2r_1 0.2r_1 10 / (2r_1 + 10 \times 0.2r_1) = r_1$$

Thus, actually, that is, with allowance for the loading by the succeeding stages, stage I and stage II have each ten (and not eleven) resistors,  $r_1$  and  $r_2$ . Then  $V_I = 0.1V_{in}$  and  $V_{II} = 0.01V_{in}$ , irrespective of the dial settings. The voltage at the divider output is then

$$V_{out} = V_I + V_2 + V_3 = 0.1V_{in}n_1 + 0.1V_I n_2 + (r/r_3) V_{II} \quad (2.100)$$

Substituting the expressions for  $V_I$  and  $V_{II}$  in (2.100) gives

$$V_{out} = (0.1n_1 + 0.01n_2 + 0.01a) V_{in} = \alpha V_{in} \quad (2.101)$$

where  $\alpha$  is the constant coefficient the value of which can be read on the dial,  $n_1$  and  $n_2$  are the outputs of decadic stages I and II ( $n_i = 0$  through 9), and  $a = r/r_3$  can be varied continuously from zero to unity.

## Chapter III

### Synthesis of Analog Differential Analyzers

#### 3.1. Development of a Block Diagram for a Differential Analyzer

**3.1.1. General.** A differential analyzer, that is, a computer intended to solve ordinary differential equations, is a mathematical model implemented with an assemblage of functional elements interconnected in a manner dictated by the ordinary differential equation (or a system of such equations) to be solved, of the form

$$a_0 \frac{dx^n}{dt^n} + a_1 \frac{dx^{n-1}}{dt^{n-1}} + \dots + a_{n-1} \frac{dx}{dt} + a_n x = f(t) \quad (3.1)$$

$$\frac{dx_i}{dt} + \sum_{j=1}^n a_{ij} x_j = b_i f_i(t) \quad (i = 1, 2, \dots, n) \quad (3.2)$$

where the coefficients  $a_0, a_1, \dots, a_n, a_{ij}$  may be constant or variable.

The dynamic behaviour of the model must be such as to be described by similar equations. The dependent and independent variables of the prototype system (mathematical variables) are related to the respective variables within the computer (machine variables) by scale factors. The solution is displayed as continually varying voltages representing the output variables. In solving differential equations, use is made of the transient response of the analogs. Therefore, the computer set-up must be arranged so that the transient times will be sufficient for accurate measurements and convenience in investigations.

In electronic differential analyzers, the main building blocks are operational amplifiers. Also, there are devices to set constant and variable coefficients, nonlinear function generators, control elements, etc.

Differential analyzers may be linear, that is, machines solving linear differential equations (such as the Soviet-made ИИТ-4, ИИТ-5 or МИИТ-11), or nonlinear, that is, machines solving nonlinear differential equations (such as the Soviet-made МН-7, МН-10, and МН-14). As will be shown in Chap. IV, differential analyzers can also solve algebraic and transcendental equations and systems. In such application, use is made of the steady-state response of the analogs, and it is sought to minimize their transient time.

**3.1.2. Developing a block diagram for the solution of ordinary differential equations.** In principle, the block diagram of a diffe-

rential analyzer may be based on two methods used in solving ordinary differential equations: by increasing and by decreasing the order of the derivative.

*Solution by increasing the order of the derivative.* Let there be a second-order linear differential equation

$$a_0 \frac{d^2 x}{dt^2} + a_1 \frac{dx}{dt} + a_2 x = f(t) \quad (3.3)$$

It may be re-written for the sought variable  $x$  as

$$x = -\frac{a_0}{a_2} \frac{d^2 x}{dt^2} - \frac{a_1}{a_2} \frac{dx}{dt} + \frac{1}{a_2} f(t) \quad (3.4)$$

As is seen, solving this equation is mainly based on the operation of differentiation. Assuming  $x$  to be known, we draw up a

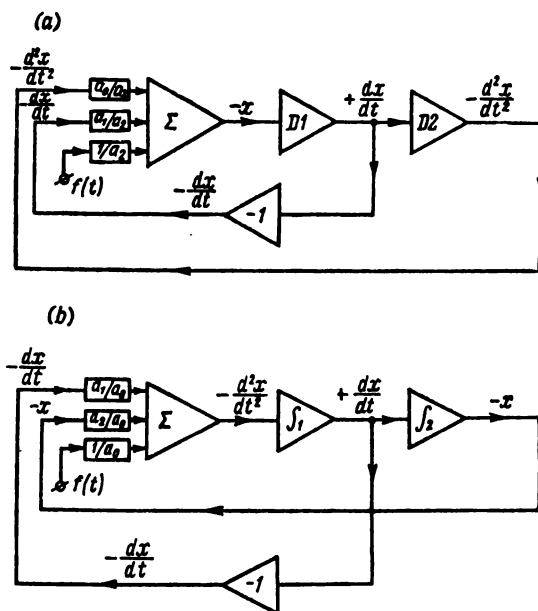


Fig. 3.1

block diagram summarizing the computer units necessary to perform the mathematical operations involved in the solution of the equation (Fig. 3.1a). From the block diagram it is seen that the computer equipment needed includes two differentiators (that is, as many as is the order of the highest derivative), a summer, a coefficient-setting unit, and an excitation,  $f(t)$ , generator.

The derivative order elevation method involving the use of differentiators is seldom used because the differentiators are sen-

sitive to input noise where the instantaneous values of the derivative are generated.

*Solution by decreasing the order of the derivative.* Let there be the same equation, Eq. (3.3). Now the equation is to be solved for the highest derivative:

$$\frac{d^2x}{dt^2} = -\frac{a_1}{a_0} \frac{dx}{dt} - \frac{a_2}{a_0} x + \frac{1}{a_0} f(t) \quad (3.5)$$

Assuming that the highest derivative is known, the sought function  $x$  can be generated by multiple integration, carrying out integration as many times as is the order of the highest derivative. Accordingly, the block diagram should include integrators (Fig. 3.1b). From the integrators, the output voltages are routed via additional computing elements (sign inverters, scalars, coefficient setting units, and the like) to a summer which together with a source of excitation,  $f(t)$ , generates the right-hand side of Eq. (3.5), that is, the highest derivative, which is applied to the first integrator.

Both methods have the same count of computing elements. Yet, the latter has found a wider use because integrators are less sensitive to noise and secure therefore a higher accuracy. Differentiators may be added to the analog computer in some special cases, such as when it is necessary to simulate lead elements in a prototype system, notably in automatic control systems.

According to the manner in which the computing elements are interconnected and problems are set up on analog computers, they may be classed into committed and uncommitted machines.

*Committed analog computers* (those with computing networks and control circuits built into the patchbay) are based on a set of first-order differential equations of the form

$$\frac{dx_1}{dt} + a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1f_1(t)$$

$$\frac{dx_2}{dt} + a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2f_2(t)$$

$$\dots \dots \dots$$

$$\frac{dx_n}{dt} + a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_nf_n(t)$$

or, in general form,

$$\frac{dx_i}{dt} + \sum_{j=1}^n a_{ij}x_j = b_if_i(t) \quad (i=1, 2, \dots, n) \quad (3.6)$$

Any differential equation or a set of equations to be solved on a committed analog computer must be reduced to the form of Eq. (3.6).

By solving each equation in the set, (3.6), for the derivative, we get

$$\frac{dx_i}{dt} = b_i f_i(t) - \sum_{j=1}^n a_{ij} x_j \quad (3.7)$$

Thus, the same block diagram applies to the computer solution of each equation in the set, (3.7).

A general block diagram for the solution of Eq. (3.7) on a committed analog computer appears in Fig. 3.2. As is seen, the computer equipment needed for the solution includes summers,

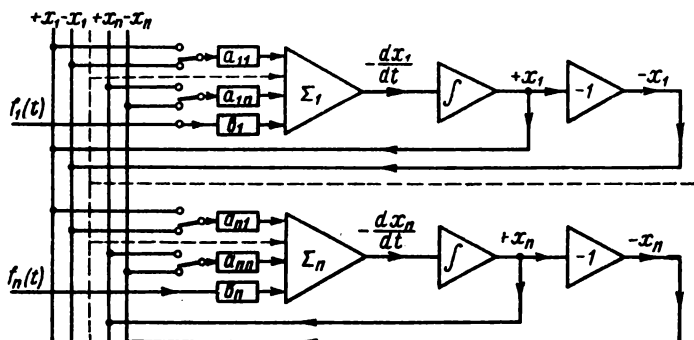


Fig. 3.2

integrators, sign inverters and devices to set the constant coefficients  $a_{ij}$  and  $b_i$ . With committed analog computers (such as the Soviet-made ИПТ-4, ЭЛИ-6, and ЭЛИ-14) a problem set-up procedure reduces to setting the coefficients of the system being integrated according to a coefficient matrix, setting initial conditions, and applying external excitations, because the computing elements are connected permanently.

A differential equation of the  $n$ th order may be reduced to the form of Eq. (3.6) in the following manner. Let there be a third-order differential equation

$$a_0 \frac{d^3 x}{dt^3} + a_1 \frac{d^2 x}{dt^2} + a_2 \frac{dx}{dt} + a_3 x = f(t) \quad (3.8)$$

By setting

$$x = x_1, \quad dx_1/dt = x_2, \quad \text{and} \quad dx_2/dt = x_3 \quad (3.9)$$

it can be seen that Eqs. (3.9) are the first equations of a system, and the third can be evaluated from Eq. (3.8), because

$$dx_3/dt = d^3 x_1/dt^3$$

As a result, Eq. (3.8) can be reduced to a set of equations of the form

$$\begin{aligned}\frac{dx_1}{dt} + 0x_1 - 1x_2 + 0x_3 &= 0f_1(t) \\ \frac{dx_2}{dt} + 0x_1 + 0x_2 - 1x_3 &= 0f_2(t)\end{aligned}\quad (3.10)$$

$$\frac{dx_3}{dt} + \frac{a_2}{a_0}x_1 + \frac{a_2}{a_0}x_2 + \frac{a_1}{a_0}x_3 = \frac{1}{a_0}f(t)$$

where

$$a_{11}=0, \quad a_{12}=1, \quad a_{13}=0, \quad b_1=0$$

$$a_{21}=0, \quad a_{22}=0, \quad a_{23}=-1, \quad b_2=0$$

$$a_{31}=a_2/a_0, \quad a_{32}=a_2/a_0, \quad a_{33}=a_1/a_0, \quad b_3=1/a_0$$

Although problem-programming on a committed analog computer is an easy matter, the machine lacks in flexibility because its elements are connected once and for all. Moreover, since such a machine contains all the elements to simulate all terms of a set of  $n$  equations, while practical problems usually involve incomplete differential equations, a sizeable proportion of computer hardware remains unused, that is, the computer contains a superfluous count of elements. This is why committed analog computers have found limited use.

*Uncommitted computers* permit changes in interconnections between otherwise not connected elements according to any equation or set of equations that may come up for solution. This principle underlies the design of the Soviet-made ИПТ-5, МПТ-9, МН-7 and МН-14 analog computers.

Uncommitted analog computers can solve equations in both their differential and integral forms. In comparison with fixed committed analog computers, they offer a wider range of values for the equation coefficients. Since the computing elements are selected and interconnected to suit a particular equation or set of equations, a higher level of utilization is achieved than with fixed-setup machines. With the same count of computer units, they can solve differential equations of higher orders. If in addition to linear elements, an uncommitted machine includes nonlinear units (such as function generators, multipliers-dividers, etc.), it will be able to handle nonlinear differential equations of various complexity. Owing to their flexibility, versatility, economy and other advantages, uncommitted analog computers are used most.

### 3.2. Main Units of Differential Analyzers

**3.2.1. Design and purpose.** As a rule, a differential analyzer essentially incorporates the units shown in the accompanying block diagram (Fig. 3.3).

(1) *The operational amplifier block, opamp*, is a combination of an operational (drift-corrected) amplifier and an assortment of input and feedback resistors and a feedback capacitor. Through an appropriate selection of input and feedback resistors, the opamp may be arranged to operate as a summer, a scaler, or a sign inverter. By switching the capacitor into the feedback path, the

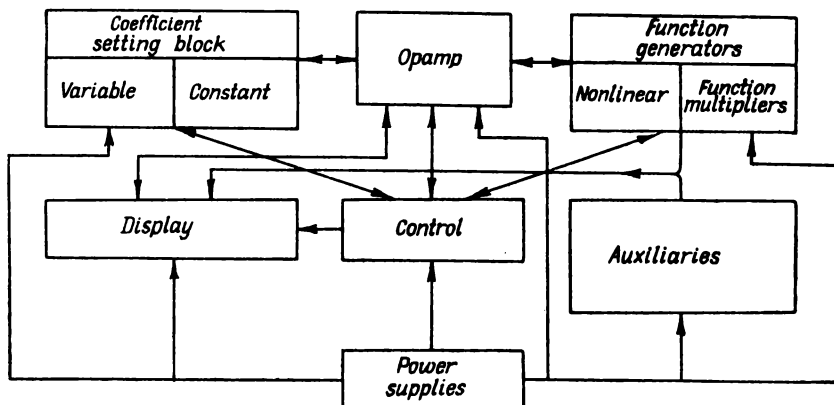


Fig. 3.3

opamp is turned into a nearly perfect integrator. The resistors and capacitor are brought in and out of circuit with toggle switches arranged on the front panel. Often, a terminal is provided on the front panel for connecting additional input resistors of other values than those included with the opamp.

Opamps may be general-purpose, that is, capable of being set to solve any of the prescribed operations, such as the opamp unit in the Soviet-made ИИТ-5 analog computer, or special-purpose, that is, settable to provide a particular operation. Examples are integrating, scaling and other opamps in the Soviet-made МИТ-9, МН-7 and МН-14 analog computers. When an opamp is set to operate as an integrator, initial conditions are introduced by a separate circuit.

(2) *The variable coefficient network* reproduces plots of coefficients by piecewise-constant (step) approximation (as has been shown in Chap. II in connection with function generators). Some machines (for example, the Soviet-made МН-7) have no such networks. The maximum value of a variable coefficient is unity.



(3) *The constant coefficient network* is usually a resistive voltage divider or potentiometer, because of which the maximum value of a constant coefficient is  $\alpha_{\max} = 1$ . Often, the network takes the form of a multi-decade potentiometer (like that discussed in Sec. 2.6). In cases where the constant coefficient must be  $\alpha > 1$ , with concurrent integration, use may be made of the expression

$$\alpha_i = k\alpha/T \quad (3.11)$$

where  $k$  = gain of the opamp

$\alpha \leq 1$  = coefficient supplied by a constant-coefficient setting network

$T = RC$  = time constant of the integrator

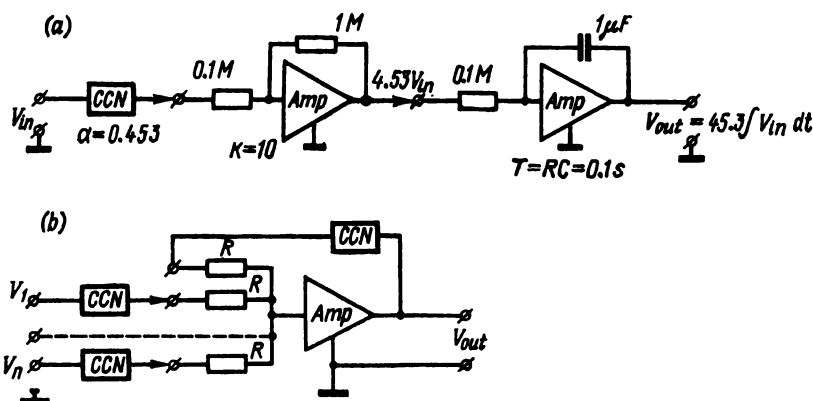


Fig. 3.4

For example, the coefficient  $\alpha_i = 45.3$  may be expressed as

$$\alpha_i = 10 \times 0.453/0.1$$

which can be generated by the circuit of Fig. 3.4a. At low values of the coefficient, this expression can be replaced with a product,  $\alpha_i = k\alpha$ , or with a quotient,  $\alpha_i = \alpha/T$ . Then the network will contain two elements, namely, a constant coefficient network and a scaler, or a constant coefficient network and an integrator. In summing amplifiers, when setting  $\alpha > 1$ , the constant coefficient network is often connected in series with the input and feedback resistors (Fig. 3.4b).

(4) *The functional elements* are included to generate functions of one or several independent variables. They may be general-purpose and special-purpose. General-purpose functional elements are available in two principal types, one, to generate a function of an independent variable (nonlinear elements) and the other,

function multipliers. An example of the former is a general-purpose diode function generator which generates functions by piecewise-linear approximation (see Sec. 2.5). A product generator is most often a quarter squares multipliers (see Sec. 2.6). As often as not, a block of nonlinear elements is built as an attachment to an analog computer. A special-purpose nonlinear element may be adapted to generate a particular function, say, a quadratic function, or to simulate dead zone, saturation, etc. Before a problem run, general-purpose nonlinear elements are set up according to the function to be generated.

(5) *The control panel*, as its name implies, provides a means to monitor and control the analog computer in operation. As a rule, the control circuit is built into a console with a panel carrying switches, knobs or buttons to balance (adjust to zero) the d. c. amplifiers and to check them for d. c. drift, to apply initial conditions and external excitations. The control circuit includes means to protect against, and actuate alarms in the case of, malfunctions. In the course of a problem run, it coordinates operation of the various units (starts the machine, terminates the solution, resets the circuits, re-runs the problem, etc.). Moreover, the control circuit provides interfacing with test equipment to monitor the progress of solution.

(6) *The display* is in effect a collection of end instruments (voltmeters, oscilloscopes, recorders and the like) the purpose of which is to present the results in a form comprehensible to the operator.

(7) *Power supplies* should meet stringent stability requirements especially for the opamps which are particularly sensitive to variations in supply voltages. As a rule, power supplies are built into self-contained units (such as the Soviet-made ЭСВ-1М and ЭСВ-6) each of which provides all the feed voltages the associated analog computer may need. For example, the ЭСВ-1М furnishes d. c. voltages of +350 V, -190 V and +75 V, and also an a. c. voltage of 6.3 V for ИПТ-4, ИПТ-5 and МПТ-9 analog computers.

**3.2.2. Operating and checking controls.** Initial conditions may be introduced in any one of two methods (Fig. 3.5).

(1) One method consists in charging the integrator capacitors to the machine values of initial conditions. This method is predominant (see Sec. 2.4). Placing switch  $S_{w2}$  to position 1 connects a resistor,  $R$ , to the input and a parallel combination of  $R_0$  and  $C$  in the feedback circuit of the integrator, thereby causing it to operate as a lag element. The capacitor accumulates a charge proportional to the initial-condition voltage. A disadvantage of this arrangement is that the initial-condition voltage,  $V_{i.c.}$ , takes some time to reach its steady-state value. The transient time can

be reduced by bringing down  $R_0$  and by placing  $C_1$  across  $R_1$ .

(2) The other method consists in that at the start of a problem run the output of each integrator is coupled to a summer and a voltage pulse is applied to its input, with an amplitude equal to the machine value of initial conditions appropriate to that integrator (Fig. 3.5b). This method sets initial conditions in a minimum time and is especially applicable to repetitive computers. However, it involves the use of an additional number of summing amplifiers.

The opamps of an analog computer are checked for d.c. drift and adjusted to zero (or balanced) by means of balancing circuits, such as shown in Fig. 3.6a. The outputs of all opamps are

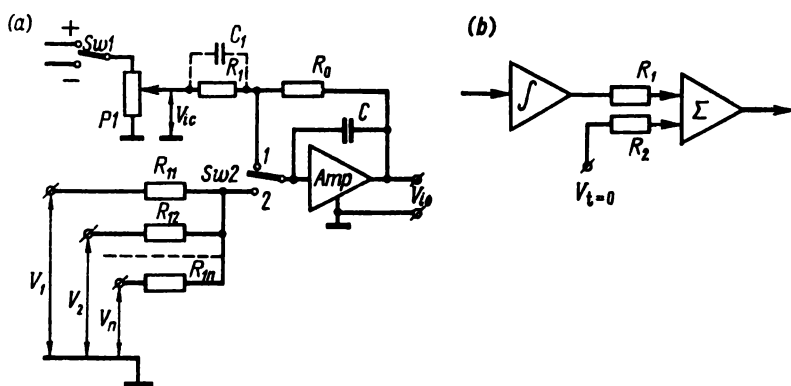


Fig. 3.5

connected to the terminals of a switch,  $Sw1$ , and this connects them in turn to a voltmeter. The voltmeter is of a multi-range variety with ranges selectable with another switch. As the first step, the output voltage of an opamp is measured on a coarse range, then on a fine one. Balancing, or zero adjustment, is done with the potentiometers which adjust the grid bias for the second stages of the d.c. amplifiers.

Should the output voltage of an opamp exceed  $\pm 100$  V because of a malfunction during a problem run, a suitable circuit (Fig. 3.6b) will turn on a neon tube to indicate an overload condition and energize a pilot relay,  $Rel_p$ . The latter will cause operation of a group of control relays, and these will disconnect the opamps from their input circuits, thereby halting the problem run until the cause of the overload is corrected.

The function generators operating by piecewise-linear approximation are set up as explained in Sec. 2.5. A plot of the desired function,  $y = f(x)$ , is approximated by straight-line segments on

the  $V_{out}$ ,  $V_{in}$  coordinates to a predetermined accuracy (Fig. 3.6c). Conversion from the  $y$ ,  $x$  to the  $V_{out}$ ,  $V_{in}$  coordinates is made on the basis of the relations

$$x = M_x V_{in}$$

$$y = M_y V_{out}$$

where  $M_x$  and  $M_y$  are the scale factors. The  $V_{out} = F(V_{in})$  plot is then presented in tabular form (Table 3.1) which gives the initial value of the function (for  $V_{in} = 0$ ), the quadrants of each straight-line segment, and the values of  $V_{in}$  and  $V_{out}$  for all node points. The quadrants in which the diode function generators

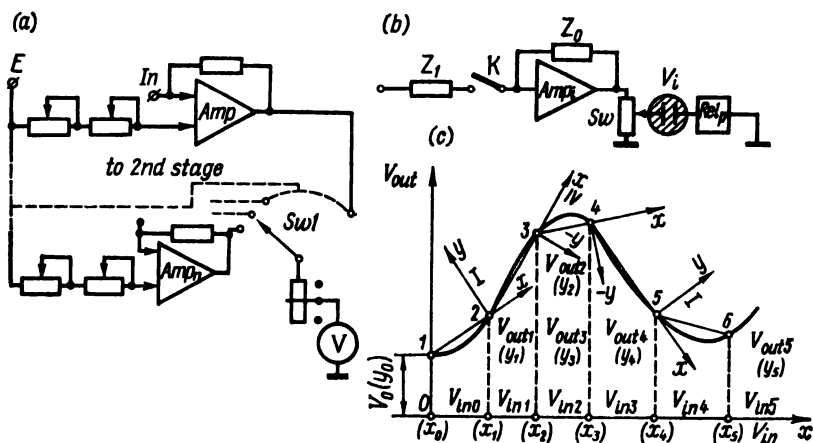


Fig. 3.6

are to operate are defined by shifting the coordinate system to node points 2, 3, 4, 5 and 6, so that the origin of coordinates is located at the beginning of a given straight-line segment, and the abscissa,  $\pm x$ , is in the direction of the preceding straight-line segment. In Fig. 3.6c, the quadrants are labelled by Roman numerals at each straight-line segment. The slope of each segment is fixed by the ordinate of the tip of each segment (that of the initial segment by the ordinate  $V_{out1}$ , that of the first by  $V_{out2}$ , etc.) and is set by adjusting  $V_{out\ i+1}$  (for each  $i$ th segment) at the generator output for each  $V_{in} = V_{in\ i+1}$ , with the aid of potentiometers.

### 3.3. Problem Preparation for a Differential Analyzer

A differential equation or a set of such equations can be prepared for solution by an electronic analog computer in any one of several ways. The methods most common at present are based

on conversion to machine equations, termwise scaling, and problem scaling by re-arrangement of the computer set-up.

Table 3.1

Function form . . . . .  $f(0) = \dots\dots\dots$

Nos.	Particulars	$kV_{in}$	Diode package, No.							
			1	2	3	4	5	6	$n-1$	$n$
1	Quadrant		I	IV	IV					
	Sign of $V_{in}$	+	+	+	+					
2	Constraint on $X$ ( $V_{in \cdot i}$ )		$V_{in1}$	$V_{in2}$	$V_{in3}$					
3a	Line segment slope (tip coordinates), $V_{in}$	$V_{in1}$	$V_{in2}$	$V_{in3}$	$V_{in4}$					
3b	Same, $V_{out}$	$V_{out1}$	$V_{out2}$	$V_{out3}$	$V_{out4}$					

The procedure involving conversion of the problem equations into machine equations is carried out in the following steps:

(1) The original system of differential equations is re-arranged to make it suitable for computer solution.

(2) The computer units required to perform the specified mathematical operations, to supply the necessary excitations and to display the desired outputs are next summarized in a block diagram which is then streamlined to obtain an optimum set-up.

(3) Machine equations are drawn up, scale factors relating each variable of the prototype system and a corresponding variable within the computer are selected, the gains of the various computer units are determined, initial conditions and other appropriate parameters of the machine are determined.

**Example.** Let the behaviour of the prototype system be described by a third-order differential equation of the form

$$a_0 \frac{d^3 x}{dt^3} + a_1 \frac{d^2 x}{dt^2} + a_2 \frac{dx}{dt} + a_3 x - f(t) = 0 \quad (3.12)$$

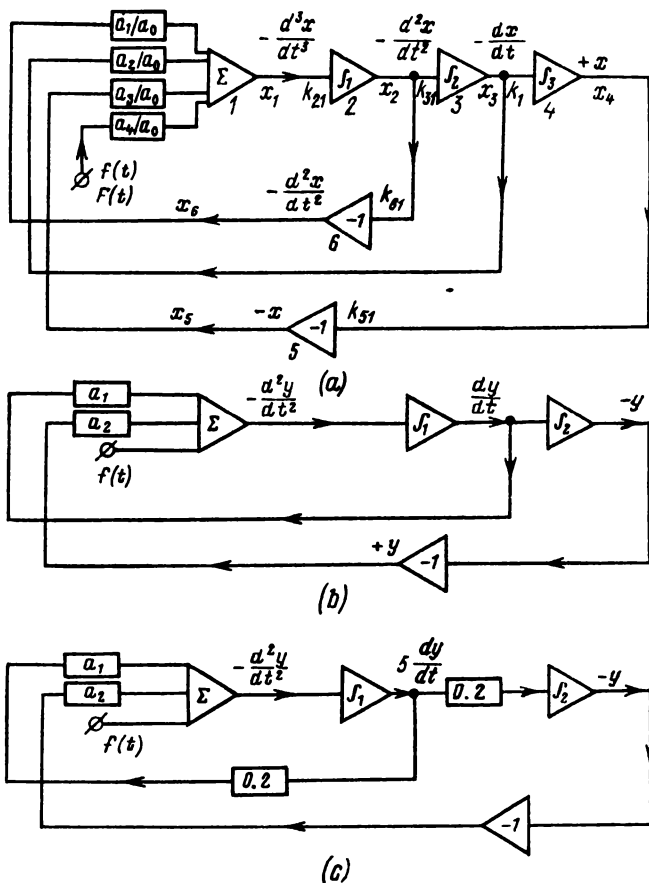


Fig. 3.7

Solving it for the highest derivative yields

$$\frac{d^3x}{dt^3} = -\frac{a_1}{a_0} \frac{d^2x}{dt^2} - \frac{a_2}{a_0} \frac{dx}{dt} - \frac{a_3}{a_0} x + \frac{1}{a_0} f(t) \quad (3.13)$$

A block diagram (Fig. 3.7) for the computer solution of this equation can be developed, using the derivative-order reduction method (in which the differential equation is solved in its integral equivalent form). The response of the computer set-up will be described by an equation analogous to Eq. (3.13), but the variables will be in the form of voltages. This will be a machine equation.

The *machine equation* analogous to the original one can be derived as follows. To begin with, an equation is written to relate

the input and output machine variables for each unit in the computer set-up (Fig. 3.7a). As a result, we obtain a set of as many equations as there are computer units in the set-up, that is

$$\begin{aligned} X_1 &= -[k_{11}X_6 + k_{12}X_3 + k_{13}X_5 + k_{15}F(\tau)] \\ X_2 &= -\frac{1}{p}k_{21}X_1, \quad X_3 = -\frac{1}{p}k_{31}X_2, \quad X_4 = -\frac{1}{p}k_{41}X_3, \\ X_5 &= -k_{51}X_4, \quad X_6 = -k_{61}X_2 \end{aligned} \quad (3.14)$$

where

$X_1, \dots, X_6$  = voltages (machine variables) at the outputs of the computer units  
 $\tau$  = machine time

$k_{11}, k_{12}, k_{13}, k_{14}, k_{21}, k_{31}, k_{41}, k_{51}, k_{61}$  = transfer functions of the computer units

The set of equations, Eq. (3.14), can be reduced to a single equation of order  $n$  (in our example, third order), written for the machine variable  $X_4$  which represents  $x$ . To do this, we express the machine variables  $X_1, X_2, X_3, X_5$  and  $X_6$  in terms of  $X_4$ :

$$\begin{aligned} X_3 &= -pX_4/k_{41} \\ X_2 &= -pX_3/k_{31} = -p^2X_4/k_{31}k_{41} \\ X_1 &= -pX_2/k_{21} = -p^3X_4/k_{21}k_{31}k_{41} \\ X_5 &= -k_{51}X_4 \\ X_6 &= -k_{61}p^2X_4/k_{31}k_{41} \end{aligned} \quad (3.15)$$

Substituting the above expressions in the first line of Eqs. (3.14) gives

$$-\frac{p^3X_4}{k_{21}k_{31}k_{41}} = -\left[-k_{11}k_{61}\frac{p^2X_4}{k_{31}k_{41}} - k_{12}\frac{pX_4}{k_{41}} - k_{13}k_{51}X_4 + k_{14}F(\tau)\right] \quad (3.16)$$

By recovering the original time functions, we obtain a machine equation of the form

$$\frac{d^3X_4}{d\tau^3} = -k_{11}k_{21}k_{61}\frac{d^2X_4}{d\tau^2} - k_{12}k_{21}k_{31}\frac{dX_4}{d\tau} - k_{13}k_{21}k_{31}k_{41}k_{51}X_4 + k_{14}k_{21}k_{31}k_{41}F(\tau) \quad (3.17)$$

The *scale factors* are found from the relations

$$M_{x_t} = x_t/X_t, \quad M_f = f(t)/F(t), \quad M_t = t/\tau$$

or

$$X_4 = x/M_{x_t}, \quad F(t) = f(t)/M_f, \quad \tau = t/M_t \quad (3.18)$$

assuming that the variables take maximum values.

The *transfer functions* of the computer units are found by comparing the machine equation, Eq. (3.17), with the original equation

ion, Eq. (3.13). Such a comparison is legitimate, however, only if both equations contain the same variables. Substituting the relations of Eq. (3.18) in Eq. (3.17) gives

$$\frac{d^3 x}{dt^3} = -\frac{k_{11}k_{21}k_{61}}{M_t} \frac{d^2 x}{dt^2} - \frac{k_{12}k_{21}k_{31}}{M_t^2} \frac{dx}{dt} - \frac{k_{13}k_{21}k_{31}k_{41}k_{61}}{M_t^3} x + \frac{k_{14}k_{21}k_{31}k_{41}M_x}{M_t M_t^3} f(t) \quad (3.19)$$

Equation (3.19) will be identical with Eq. (3.13) on the condition that

$$k_{11}k_{21}k_{61}/M_t = a_1/a_0; \quad k_{12}k_{21}k_{31}/M_t^2 = a_2/a_0 \\ k_{13}k_{21}k_{31}k_{41}k_{51}/M_t^3 = a_3/a_0; \quad k_{14}k_{21}k_{31}k_{41}M_x/M_t M_t^3 = 1/a_0 \quad (3.20)$$

Because in Eq. (3.20) the number of unknown transfer functions exceeds that of equations, some of the transfer functions are chosen arbitrarily from other considerations, and the remainder from Eqs. (3.20). The transfer functions for some of the various opamp connections are as follows:

$k_t = -1/pRC$  for an opamp connected as an integrator

$k_d = -pRC$  for an opamp connected as a differentiator

$k_s = -R_0/R$  for an opamp connected as a scaler

$k_{inv} = -1$  for an opamp connected as a sign inverter

Using the appropriate transfer function, all the circuit parameters can readily be found by a simple calculation. For example, if the second computer unit is to operate as an integrator, its transfer function will be

$$k_{21} = -1/pRC$$

Then, assuming the transfer function to be equal to unity and choosing  $C = 1 \mu F = 10^{-6}$  F, the input resistor  $R$  will be  $10^6$  ohms. A similar procedure can be followed in finding circuit parameters for other computer units.

*The machine initial conditions and external excitations are calculated on the basis of the scaling equations, Eqs. (3.18). Let, in our example, the initial conditions and external excitations be specified as*

$$(dx/dt)_0 = (d^2x/dt^2)_0 = 0$$

$$x_0 = 1$$

$$f(t)_0 = \text{constant} = C$$

Using Eqs. (3.18), we obtain

$$X_4(0) = x_0/M_x$$

$$F(\tau)_0 = f(t)_0/M_f = C/M_f$$



Consider a modification of the above procedure. Let the prototype system now be described by a set of linear differential equations of the form

$$\begin{aligned} dx_1/dt &= a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + b_1f_1(t) \\ dx_2/dt &= a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + b_2f_2(t) \\ dx_3/dt &= a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + b_3f_3(t) \end{aligned} \quad (3.21)$$

subject to initial conditions

$$x_1(0) = a_1, \quad x_2(0) = a_2, \quad x_3(0) = a_3 \quad (3.22)$$

The scale factors relating the mathematical and machine variables are

$$M_{x_i} = x_i/X_i, \quad M_{f_i} = f_i(t)/F_i(\tau), \quad M_t = t/\tau \quad (3.23)$$

where  $i = 1, 2, 3$ .

Substituting the expressions for  $x_i$  and  $f_i(t)$  from (3.23) into (3.21) and re-writing each equation for the derivatives  $dX_i/dt$ , we obtain a set of machine differential equations (for  $t = \tau$  and  $M_t = 1$ ):

$$\begin{aligned} dX_1/d\tau &= a_{11}X_1 + a_{12} \frac{M_{x_2}}{M_{x_1}} X_2 + a_{13} \frac{M_{x_3}}{M_{x_1}} X_3 + b_1 \frac{M_{f_1}}{M_{x_1}} F_1(\tau) \\ dX_2/d\tau &= a_{21} \frac{M_{x_1}}{M_{x_2}} X_1 + a_{22}X_2 + a_{23} \frac{M_{x_3}}{M_{x_2}} X_3 + b_2 \frac{M_{f_2}}{M_{x_2}} F_2(\tau) \\ dX_3/d\tau &= a_{31} \frac{M_{x_1}}{M_{x_3}} X_1 + a_{32} \frac{M_{x_2}}{M_{x_3}} X_2 + a_{33}X_3 + b_3 \frac{M_{f_3}}{M_{x_3}} F_3(\tau) \end{aligned} \quad (3.24)$$

If  $t \neq \tau$ , the coefficients of  $X_i$  and  $F_i(\tau)$  will include  $M_t$ . The machine values of initial conditions are

$$X_i(0) = a_i/M_{x_i} \quad (i = 1, 2, 3)$$

The scale factors are chosen to satisfy the conditions

$$\begin{aligned} M_{x_i} &\geq |x_i|_{\max}/X_i \max \\ M_{f_i} &\geq |f_i(t)|_{\max}/F_i(\tau) \max \end{aligned} \quad (3.25)$$

By substituting the expressions for  $M_{x_i}$  and  $M_{f_i}$  in (3.24), we find the numerical values of the constant coefficients  $A_{ij}$  of the machine variables. As a result, the set of machine equations, Eqs. (3.24), can now be given the form

$$\begin{aligned} dX_1/d\tau &= A_{11}X_1 + A_{12}X_2 + A_{13}X_3 + B_1F_1(\tau) \\ dX_2/d\tau &= A_{21}X_1 + A_{22}X_2 + A_{23}X_3 + B_2F_2(\tau) \\ dX_3/d\tau &= A_{31}X_1 + A_{32}X_2 + A_{33}X_3 + B_3F_3(\tau) \end{aligned} \quad (3.26)$$

where  $A_{11} = a_{11}$ ,  $A_{22} = a_{22}$ ,  $A_{33} = a_{33}$ ,  $A_{ij} = a_{ij} (M_{x_j} / M_{x_i})$ ,  $B_i = b_i (M_{f_i} / M_{x_i})$  ( $i = 1, 2, 3$ ).

Finally a block diagram for the computer solution of Eq. (3.26) is developed.

The procedures for problem preparation and scaling discussed above become rather cumbersome when it comes to systems of nonlinear differential equations. Nor do they allow transfer functions for the various computer units to be chosen with allowance for the range of changes in the intermediate variables. Furthermore, the operator cannot directly monitor the behaviour of the prototype system because the computer solves a machine equation which is superficially unrecognizable as an alternative formulation of the original problem. This is why the procedures presented above are limited to relatively simple differential equations and systems.

With termwise scaling, the scale factors and transfer functions are found separately for each of the computer units. Thus, the integral

$$z = k \int_0^t x dt$$

is mechanized by an integrating opamp for which the input-output relation has the form

$$V_{out} = (1/RC) \int_0^t V_{in} dt$$

Using the appropriate scale factors, the problem and machine variables may be related as

$$V_{in} = x/M_x$$

$$V_{out} = z/M_z$$

Substituting these expressions for  $V_{in}$  and  $V_{out}$  in the previous relation and comparing the resultant expression with  $z = f(t)$ , we obtain

$$M_z = RC (M_x/k)$$

On the basis of this relation, we can determine the time constant  $RC$  or one of the scale factors,  $M_x$  or  $M_z$ , if the other two terms are known or chosen from some other considerations.

A similar procedure may be applied to any other computer units. Sometimes, conversion to machine equations and termwise scaling are used simultaneously.

Problem preparation based on problem scaling through re-arrangements in the computer block diagram consists in the fol-

lowing. Let the prototype system be described by a differential equation of the form

$$d^2y/dt^2 + a_1 dy/dt + a_2y = f(t)$$

The initial conditions are

$$y = y_0$$

$$dy/dt = (dy/dt)_0$$

The first step is to draw up a block diagram for computer solution on the assumption that all scale factors are equal to unity (see Fig. 3.7*b*). As is seen, there are two loops, namely:

- (1) a summer, the first integrator, and a scaler;
- (2) a summer, two integrators, a sign inverter, and a scaler.

At any point on these loops, the scale factors can be adjusted to any desired value, provided the loop gain remains unchanged. For example, if the output of the first integrator need be  $5dy/dt$ , the gain at all succeeding points on both loops must be set to 0.2. Then the block diagram will be modified to take the form shown in Fig. 3.7*c*. The initial conditions will then be

$$y = y_0$$

$$dy/dt = 5(dy/dt)_0$$

### 3.4. Problem Solving on an Electronic Differential Analyzer

**3.4.1. General steps in problem solving.** In brief, the steps involved in problem solving on a differential analyzer are as follows:

- (1) The problem is set up on the machine. The gains are adjusted for the various computer units.
- (2) The problem is given a check run. The range of output voltage is determined for each computer unit. The scale factors and gains are adjusted as may be required.
- (3) The problem solution is verified.
- (4) The data comprising the solution of the problem are recorded and processed.

A problem is set up on a computer according to the block diagram as follows:

- (a) the opamps specified in the block diagram are selected and connected to handle appropriate operations;
- (b) the opamps thus selected are checked for d.c. drift and balanced, if necessary;
- (c) the variable-coefficient potentiometers are set;
- (d) the constant coefficients are set;
- (e) the computer units are interconnected according to the computer set-up diagram by means of plugs and cords either on a patch board (as is done on the Soviet-made MH-7 and MIT-9

analog computers) or directly through sockets on the front panels of the units;

(f) external excitations are applied as voltages from independent sources. These voltages should be identical with the actual excitation in the physical problem in waveform, amplitude, frequency and phase;

(g) initial conditions are applied to the integrators as explained in Sec. 3.2. The necessary connections are made at one terminal only (the other is automatically returned to ground). The numerical values of the initial voltages are set with a voltmeter;

(h) the computer is started by pressing the START button on the control panel. In the course of the check run, the output voltages of the various units and of the computer as a whole are measured or recorded, and the scale factors and gains are revised and improved as may be necessary. Matters related to the verification of the problem solution will be taken up later.

**3.4.2. Function generation by solution of auxiliary differential equations.** A variety of functions may be mechanized with standard computing units (integrators, summers, scalers and sign inverters) by integrating auxiliary differential equations.

The method boils down to the following. A given function is differentiated repeatedly until a differential equation is derived whose solution is the specified function and which may be implemented without any special-purpose function generators. The equation thus derived is called the auxiliary (or characteristic) equation. Then a block diagram is drawn up for the computer solution of the auxiliary differential equation. All the operations involved in this solution (integration, addition, scaling, or sign inversion) can be implemented with a complement of standard opamps connected as integrators, summers, scalers and sign inverters.

**Example 1.** A function,  $x = \sin \omega t$ , is to be implemented. By differentiating this function, we get

$$dx/dt = \omega \cos \omega t$$

This is not a characteristic equation because a function generator is needed to implement the function  $\cos \omega t$ . By differentiating again, we obtain

$$d^2x/dt^2 = -\omega^2 \sin \omega t = -\omega^2 x$$

or

$$d^2x/dt^2 + \omega^2 x = 0 \quad (3.27)$$

Equation (3.27) is characteristic, because under initial conditions

$$t=0, \quad x_0=0, \quad (dx/dt)_0=\omega$$

its solution is the specified function. The relevant computer set-up for solution of Eq. (3.27) appears in Fig. 3.8a.

**Example 2.** The desired function is specified as a polynomial of degree three:

$$x = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \quad (3.28)$$

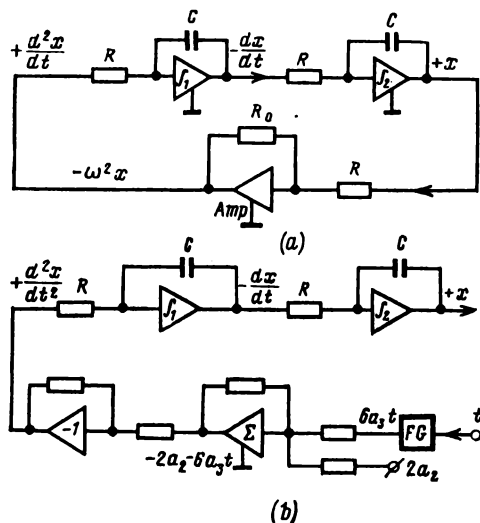


Fig. 3.8

By differentiating Eq. (3.28) twice, we obtain the characteristic equation

$$d^2x/dt^2 = 2a_2 + 6a_3t \quad (3.29)$$

whose solution under initial conditions

$$t = 0, \quad x = a_0, \quad dx/dt = a_1$$

is the polynomial, Eq. (3.28). The computer set-up for solution of Eq. (3.29) is given in Fig. 3.8b.

**3.4.3. Solution of nonlinear equations on electronic analog computers.** Nonlinear differential equations occur in many fields of science and technology, notably in the study of automatic control systems, hypersonic flight (where nonlinearity in aerodynamic characteristics must be taken into account), thermal engineering, etc. Since a rigorous theory for solving nonlinear problems is still lacking, their computer solution gains special importance.

As with solving linear differential equations on a computer, the general steps as regards nonlinear equations are re-arrangement of the nonlinear differential equation into a form suitable for

computer solution, setting up the problem on the machine, problem solution, and solution readout.

Preparation of a nonlinear differential equation is carried out to a similar program as with linear equations, but it includes some additional steps because of the need for function generators to simulate the specified nonlinearities. Nonlinear functions may be generated in any one of the ways explained earlier, namely by solving auxiliary differential equations, through the use of general-purpose function and product generators, or with a complement of standard highly specialized nonlinear elements.

**3.4.4. Accuracy of solution and solution checks.** As already noted, the accuracy of analog computers is limited and depends on a number of factors, such as spread in circuit parameters, inaccuracies in assembly, wiring or measurements, external influences, etc. Many of the errors are random, which fact necessitates application of probability theory to error analysis.

Errors may affect an analog computer in several ways. In some cases errors may bring about a qualitative change in problem solving; in others, they may affect reproducibility (repeatability of solution) if the computer is sensitive to even small changes in initial conditions and external excitations. These difficulties increase still more when an analog computer is used for a study into error-sensitive engineering systems.

To obtain the utmost in accuracy from a given analog computer, it is essential, before a problem is set up, to revise and improve the computer set-up in order to minimize the number of units used, to check and adjust all opamps for d.c. drift, and to verify that all computing units do the mathematical operations prescribed.

After a problem has been set up on the computer, a d. c. drift check must be applied to the whole set-up. In a fairly simple manner, this can be done as follows: (a) with a system of homogeneous differential equations, the equations are set up, but initial conditions are not applied; the solution must be zero; (b) with a system of nonhomogeneous differential equations, initial conditions are not applied and sources of external excitations are disconnected; the solution must likewise be zero.

Practically, a zero solution will not be obtained in each of the two cases, but the deviation from zero must not exceed a few tenths of a volt.

If a problem is stated as a system of linear differential equations of the form

$$\begin{aligned} dx_1/dt &= f_1(x_1, x_2, \dots, x_n) \\ dx_2/dt &= f_2(x_1, x_2, \dots, x_n) \\ &\vdots \\ dx_n/dt &= f_n(x_1, x_2, \dots, x_n) \end{aligned} \quad (3.30)$$

subject to initial conditions

$$x_1(0) = x_{10}, \quad x_2(0) = x_{20}, \quad \dots, \quad x_n(0) = x_{n0}$$

the accuracy of problem solving may be checked as follows. To begin with, instead of the specified equations, one solves an auxiliary system of the form

$$\begin{aligned} dx_1/dt &= f_1(x_1, x_2, \dots, x_n) - f_1(x_{10}, x_{20}, \dots, x_{n0}) \\ dx_2/dt &= f_2(x_1, x_2, \dots, x_n) - f_2(x_{10}, x_{20}, \dots, x_{n0}) \\ \vdots \\ dx_n/dt &= f_n(x_1, x_2, \dots, x_n) - f_n(x_{10}, x_{20}, \dots, x_{n0}) \end{aligned} \quad (3.31)$$

whose solution is

$$x_1 = x_{10}, \quad x_2 = x_{20}, \quad \dots, \quad x_n = x_{n0}$$

The amount by which the solution differs from the above values will give the degree of accuracy obtained.

The accuracy of the solution can be assessed through analysis of the error equations, but the practical realization of this procedure is far from easy. Ordinarily, resort is made to a number of indirect methods of error estimation, such as repeated runs of the same problem and a comparison of the solutions obtained on the same computer, a comparison of the solutions from several computers, a comparison of computer solution with a theoretical one, solving of test problems, etc.

## Algebraic and Transcendental Equation Solvers

**4.1.1. General.** Any algebraic equation may be written as a polynomial of degree  $n$ :

**or**

$$\sum_{i=0}^n a_i x^i = 0$$

Transcendental equations (that is, those which involve non-algebraic operations on the unknown variable, such as taking a logarithm, finding trigonometric functions, etc.) are likewise often met with in practice.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n - b_1 &= 0 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n - b_2 &= 0 \end{aligned} \quad (4.2)$$

$$\overset{\cdot}{a}_{n1}\overset{\cdot}{x}_1 + \overset{\cdot}{a}_{n2}\overset{\cdot}{x}_2 + \dots + \overset{\cdot}{a}_{nn}\overset{\cdot}{x}_n - \overset{\cdot}{b}_n = 0$$

or

$$\sum_{j=1}^n a_{ij}x_j - b_i = 0 \quad (i=1, 2, \dots, n) \quad (4.3)$$

Algebraic and transcendental equations and their systems may be solved on electronic simulators and on special-purpose equation solvers based on a variety of methods (root-adjustment, scanning, iteration, etc.).

According to the roots sought, the special-purpose equation solvers for algebraic and transcendental equations may be classed into two groups as follows:

- (a) **adjuster-type equation solvers** which find one or several roots of an equation or a set of equations;
- (b) **polynomial evaluators** which find real and complex roots for algebraic equations and their systems.

**4.1.2. Adjuster-type equation solvers.** With this method, which calls for a very simple computer set-up, the equation (or the



system of equations) to be solved is specified implicitly

$$f(x) = 0 \quad (4.4)$$

$$F_i(x_1, x_2, \dots, x_n) = 0 \quad (i = 1, 2, \dots, n) \quad (4.5)$$

The procedure consists in trying several values of  $x$  consecutively until a value is found such that the function  $f(x)$  vanishes. The computer set-up must ensure rapid convergence and short transients.

The circuit of a practical adjuster-type equation solver is shown in Fig. 4.1. Initially, the function generator,  $FG$ , accepts an arbitrary value of  $x$  which is different from the root of the equation.

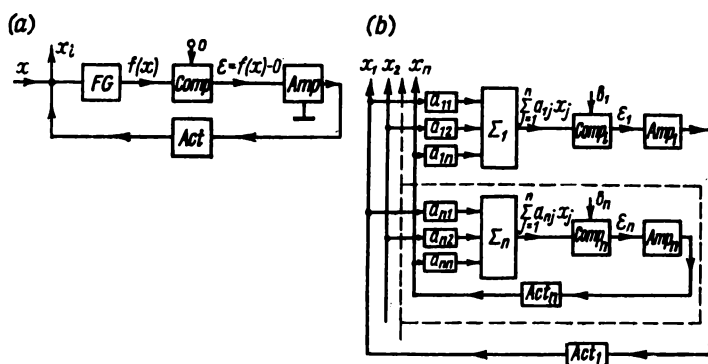


Fig. 4.1

In response, the function generator generates a function,  $f(x)$ . A comparator,  $Comp$ , matches it with zero and generates an error signal,  $\varepsilon = f(x) - 0$ . Boosted in power and voltage by an amplifier,  $Amp$ , the error signal is applied to an actuator,  $Act$  (say, a servo motor) which operates according to the sign of the error signal so as to change the value of  $x$  and minimize the error signal to zero,  $\varepsilon = f(x) - 0$ . The value of  $x$  existing at the input to the function generator at that instant will be the root of the equation being solved.

Some of the elements shown in the diagram of Fig. 4.1 may be omitted, according to the physical quantities chosen to represent  $x$  and  $\varepsilon$ . For example, there may be neither a comparator nor an amplifier, if  $x$  is a voltage and if the signal  $f(x)$  carries enough power.

Figure 4.1b is a block diagram of a servo-operated feedback equation solver designed to solve systems of linear algebraic equations of the form of Eq. (4.2). When arbitrary values of  $x_1, x_2, \dots, x_n$  are set in the solver, the comparators,  $Comp_i$ , form

difference signals,  $e_i = \sum_{j=1}^n a_{ij}x_j - b_i$ , which are applied to a closed-loop servo, and this operates to balance the circuit ( $e_1 = e_2 = \dots = e_n = 0$ ). The steady-state values of the  $x_i$ 's ( $i = 1, 2, \dots, n$ ) will be the roots of the system. The comparators and the actuators may be interconnected in a variety of ways. With the interconnections shown in Fig. 4.1b, a parallel-feedback solver results. If, on the other hand, *Comp*<sub>1</sub> is connected to *Act*<sub>2</sub>, and *Comp*<sub>2</sub> to *Act*<sub>1</sub>, a cross-coupled solver is obtained. A form of interconnections is chosen so as to secure a maximum sensitivity of the solver to feedback and to satisfy the stability criterion.

An equation solver for a system of nonlinear algebraic and transcendental equations is built along similar lines but it is extended to include function generators to implement nonlinear functions.

**4.1.3. Polynomial evaluators.** These analog computers are designed to solve algebraic equations of the form of Eq. (4.1) which may, in the general case, have both real and complex coefficients  $a_i$  and roots  $x_i$ .

The solution of an algebraic equation obtained as a complex number may be written in algebraic, exponential or trigonometric form:

$$x = \alpha + j\beta \quad (4.6)$$

$$x = \rho \exp(j\theta) \quad (4.7)$$

$$x = \rho(\cos \theta + j \sin \theta) \quad (4.8)$$

The polynomial evaluator operates to find the values of  $\alpha$  and  $\beta$  (or  $\rho$  and  $\theta$ ) for the root  $x$  that will satisfy the polynomial, Eq. (4.1). Since any system of algebraic equations can always be reduced to a single algebraic equation of degree at most  $n = n_1 n_2 \dots n_n$ , where the  $n_i$ 's are the degrees of the equations in the system ( $i = 1, 2, \dots, n$ ), polynomial evaluators are usually set up to solve a single equation.

The root-searching procedure consists in transforming the polynomial, Eq. (4.1), by substituting (4.6), (4.7), (4.8) for the variable  $x$ . As a result, a complex equation of the form

$$A + jB = 0 \quad (4.9)$$

is obtained where  $A$  is the real part and  $B$  is the imaginary part which is a function of  $\alpha$  and  $\beta$  or of  $\rho$  and  $\theta$ . The value of  $x$  will be a root of the equation, Eq. (4.1), if it causes the real and imaginary parts of (4.9) to vanish simultaneously, that is,

$$\begin{aligned} A &= 0 \\ B &= 0 \end{aligned} \quad (4.10)$$

The system of equations, Eq. (4.10), can be implemented with a set-up similar to that shown in Fig. 4.2. A computing unit, *CU*, when fed the values of coordinates from a coordinate-setting unit, *CSU*, and the coefficients  $a_i$ , evaluates the real and imaginary parts,  $A$  and  $B$ , of the function, and these are plotted one

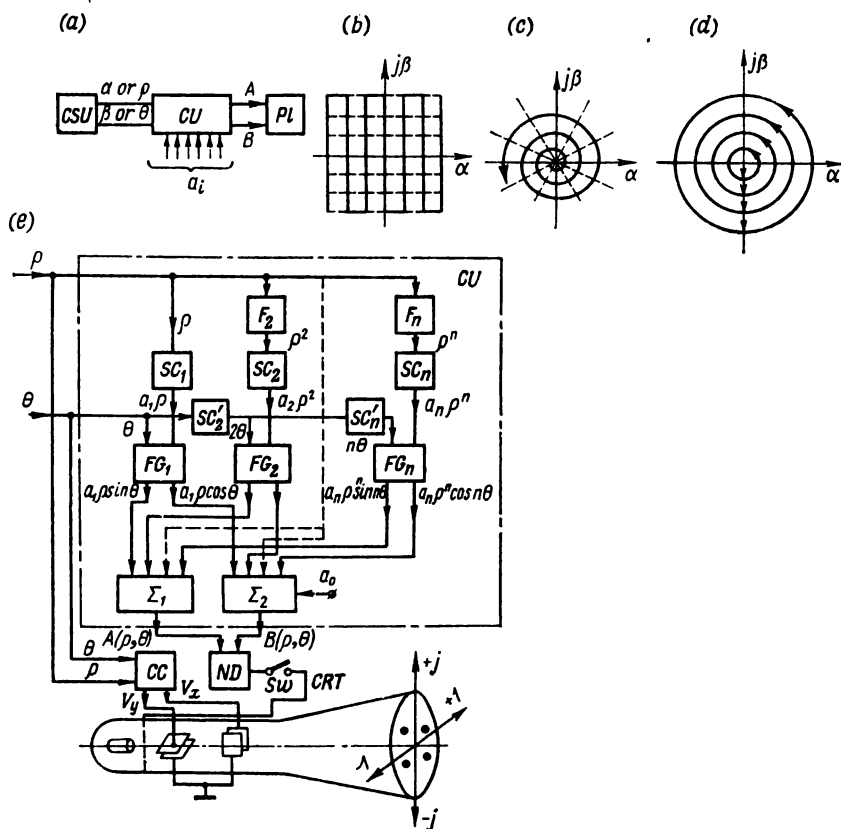


Fig. 4.2

against the other on a plotter,  $P$ , which shows when a zero of the polynomial,  $A = 0$ ,  $B = 0$ , has been found. The set-up should be an open-loop one, because with a closed-loop (feedback) arrangement, the coefficients  $a_i$  tend to upset the stability of the system. The coordinates may be set manually or automatically, and the roots are determined by the scanning technique. In the scanning technique, the computer sweeps the complex plane by

varying the  $x_i$ 's in a predetermined manner and locates the points where the polynomial vanishes.

Among the scanning patterns to be considered here are linear scanning (Fig. 4.2b), spiral scanning (Fig. 4.2c) and circular scanning (Fig. 4.2d). In linear scanning the complex plane is swept along horizontal and vertical straight lines. In spiral scanning the path is a spiral (if  $\omega_p \ll \omega_\theta$ ) or radials (if  $\omega_p \gg \omega_\theta$ ). In circular scanning,  $\rho$  is held constant during each scan and only the angle  $\theta$  is varied, so that a family of concentric circles is obtained, which bracket the individual roots. By narrowing the brackets, the roots may be isolated.

In most cases, polynomial evaluators utilize the trigonometric presentation of (4.8). Substituting it in (4.1) gives

$$a_n \rho^n (\cos \theta + j \sin \theta)^n + a_{n-1} \rho^{n-1} (\cos \theta + j \sin \theta)^{n-1} + \dots \\ \dots + a_1 \rho (\cos \theta + j \sin \theta) + a_0 = 0 \quad (4.11)$$

or

$$[a_n \rho^n \cos n\theta + a_{n-1} \rho^{n-1} \cos (n-1)\theta + \dots + a_0] \\ + j [a_n \rho^n \sin n\theta + a_{n-1} \rho^{n-1} \sin (n-1)\theta + \dots + a_1 \rho \sin \theta] = 0$$

In a more concise form,

$$\sum_{i=0}^n a_i \rho^i \cos i\theta + j \sum_{i=0}^n a_i \rho^i \sin i\theta = 0$$

or

$$A + jB = 0$$

that is, we have obtained Eq. (4.9) in which

$$A = F_1(\rho, \theta) = \sum_{i=0}^n a_i \rho^i \cos i\theta$$

$$B = F_2(\rho, \theta) = \sum_{i=0}^n a_i \rho^i \sin i\theta$$

By equating  $A$  and  $B$  to zero, we obtain a system of equations of the form of Eq. (4.10):

$$\begin{aligned} F_1(\rho, \theta) &= 0 \\ F_2(\rho, \theta) &= 0 \end{aligned} \quad (4.12)$$

The above system, Eq. (4.12), can be implemented with a polynomial evaluator built around a cathode-ray tube as shown in the block diagram of Fig. 4.2d. An analog computer, *Comp*, incorporating function generators  $F_i$  and  $FG_i$ , scalars  $SC_i$  and  $SC'_i$ , and summers  $\Sigma_1$  and  $\Sigma_2$ , accepts the coefficients  $a_i$  and the coordinates  $\rho$  and  $\theta$  and generates the functions  $F_1(\rho, \theta)$  and  $F_2(\rho, \theta)$ . The flow of computation leading to  $A$  and  $B$  is clear

from reference to the block diagram. The coordinate converter, *CC*, converts the polar ( $\rho, \theta$ ) coordinates to the cartesian ( $V_x, V_y$ ) coordinates, which are applied to the horizontal and vertical deflection plates of the *CRT*, causing the electron beam to sweep the complex plane (the *CRT* screen) in a predetermined manner. The null detector, *ND*, senses the deviations of the functions *A* and *B* from zero and controls the switch *Sw* appropriately. As long as  $A \neq 0$  and  $B \neq 0$ , the switch *Sw* is closed, an inhibit signal is applied to the *CRT* grid. At the instants when  $A = 0$  and  $B = 0$ , the switch opens, the inhibit signal is removed, and the beam reaches the screen to produce a light spot on it. The location of the light spot defines the root of the polynomial. The values of  $\rho$  and  $\theta$  that satisfy the equation may be noted in several ways. One is to note the settings of the elements that set  $\rho$  and  $\theta$  at the instants when the functions *A* and *B* vanish simultaneously. With a *CRT* screen of long afterglow, the operator can conveniently watch the location of, and changes in, the roots. This feature is of special value in the analysis of automatic control systems for the purpose of system evaluation and optimization.

## 4.2. Techniques for Solving Linear Algebraic Equations on Electronic Analog Computers

**4.2.1. Solution by root adjustment.** Let there be a system of equations of the form of (4.2)

$$\begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n - b_1 & = & 0 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n - b_2 & = & 0 \\ \vdots & & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n - b_n & = & 0 \end{array}$$

which has a unique solution if the determinant  $D \neq 0$ . It is assumed that the coefficients  $a_{ii}$  ( $i = 1, 2, \dots, n$ ) are non-zero and greater than other coefficients in magnitude. To set up the system on the computer, it is transformed as

$$\begin{aligned} x_1 &= d_{11}x_1 + d_{12}x_2 + \dots + d_{1n}x_n + e_1 \\ x_2 &= d_{21}x_1 + d_{22}x_2 + \dots + d_{2n}x_n + e_2 \\ &\vdots \\ x_n &= d_{n1}x_1 + d_{n2}x_2 + \dots + d_{nn}x_n + e_n \end{aligned} \quad (4.13)$$

where  $d_{11} = d_{22} = \dots = d_{nn} = 0$

$$d_{ij} = -\frac{a_{ij}}{a_{ii}} \text{ (} i \text{ is the No. of an equation}$$

and  $j$  is the No. of a term  
in each equation)

The problem is then solved by adjusting the  $x_i$ 's so as to satisfy Eq. (4.13). In solving Eq. (4.13), the computer will be stable if it contains an odd number of operational amplifiers. This will be the case if all coefficients of the variables above the principal diagonal are positive and those below the diagonal are negative.

A computer set-up with an odd number of opamps to solve a system of three equations of the form

$$\begin{aligned}x_1 &= d_{11}x_1 - d_{12}x_2 - d_{13}x_3 + e_1 \\x_2 &= +d_{21}x_1 + d_{22}x_2 - d_{23}x_3 + e_2 \\x_3 &= +d_{31}x_1 + d_{32}x_2 + d_{33}x_3 + e_3\end{aligned}$$

is shown in Fig. 4.3. Here,  $d_{11} = d_{22} = d_{33} = 0$ .

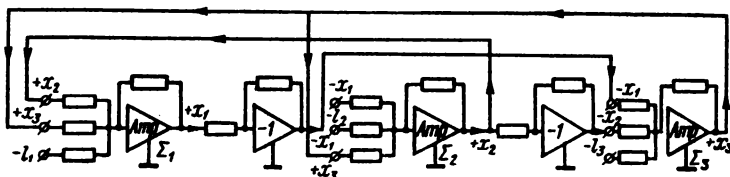


Fig. 4.3

If the gains over all closed loops of the computer set-up are less than unity, an even number of opamps may be used.

**4.2.2. Solution by transformation into a set of differential equations (the characteristic equation method).** The given set of algebraic equations

$$\sum_{j=1}^n a_{ij}x_j - b_i = 0 \quad (i = 1, 2, \dots, n) \quad (4.14)$$

is transformed into a set of ordinary differential (characteristic) equations of the form

$$\sum_{j=1}^n a_{ij}x_j - b_i = dx_i/dt \quad (4.15)$$

The process will be a damped one, and the system will be stable if the derivatives tend to zero for  $t \rightarrow \infty$  that is, if

$$\lim_{t \rightarrow \infty} x_i(t) = x_i$$

The stability criterion is satisfied if all roots of the respective characteristic equation have negative real parts. In other words, after all transients die out, the steady-state values of the variables  $x_i$  satisfy Eq. (4.14), and it can be solved by the set-up for

Eq. (4.15), in the ordinary manner. Otherwise the computer set-up yields unstable transient solutions  $x_1(t)$ ,  $x_2(t)$ , ...,  $x_n(t)$ .

If it is not known in advance which sign the real parts of the roots of the characteristic equation will have, special methods may have to be employed in transforming the system, Eq. (4.15), so as to obtain stable solutions.

**4.2.3. Iteration (successive-approximation) method.** The simple iteration method consists in that the system, Eq. (4.2), is transformed into a set, Eq. (4.13). Starting with arbitrary values,  $x_i^{(0)}$ , for the variables and substituting them in Eq. (4.13), we obtain the first approximation

$$\begin{aligned}x_1^{(1)} &= d_{12}x_2^{(0)} + d_{13}x_3^{(0)} + \dots + d_{1n}x_n^{(0)} + e_1 \\x_2^{(1)} &= d_{21}x_1^{(0)} + d_{23}x_3^{(0)} + \dots + d_{2n}x_n^{(0)} + e_2 \\&\vdots \\x_n^{(1)} &= d_{n1}x_1^{(0)} + d_{n2}x_2^{(0)} + \dots + d_{n(n-1)}x_{n-1}^{(0)} + e_n\end{aligned}\quad (4.16)$$

The values,  $x_i^{(1)}$ , thus found are again substituted in the system, Eq. (4.13), to give a second approximation, etc. The process is repeated until the necessary degree of accuracy is obtained. The process will converge if the system, Eq. (4.2), satisfies one of the following inequalities

$$\begin{aligned}\sum_{j=1}^n \left| \frac{a_{ij}}{a_{ii}} \right| &< 1 \\ \sum_{i=1}^n \left| \frac{a_{ij}}{a_{ii}} \right| &< 1 \quad \sum_{i,j=1}^n \left( \frac{a_{ij}}{a_{ii}} \right)^2 < 1\end{aligned}\quad (4.17)$$

in which all terms vanish at  $i = j$ . The computer set-up suitable to implement simple iteration is the same as for solving by root adjustment.

A more refined technique is the Gauss-Seidel iteration method. With this method, a system of algebraic equations is written as

$$\sum_{j=1}^n a_{ij}x_j - b_i = e_i \quad (i = 1, 2, \dots, n) \quad (4.18)$$

For the values of the  $x_i$ 's which are not the roots of the equation

$$\sum_{j=1}^n a_{ij}x_j \neq b_i, \quad e_i \neq 0$$

The left-hand side of the computer set-up shown in Fig. 4.4 has implemented the system, Eq. (4.18), if the null detectors,  $ND_i$ , read zero, that is. if  $e_1 = e_2 = \dots = e_n = 0$ . At that instant,

the system (4.18) reduces to (4.3), and the values of  $x_1, x_2, \dots, x_n$  will be the roots of the equations.

**4.2.4. Minimization method.** Instead of adjusting one variable to make the error in one equation zero, as is done in the Gauss-Seidel method, the minimization method requires that all variables be adjusted simultaneously so that the sum of the absolute values of all errors,  $e_i$ , be reduced to zero. For arbitrary values  $x_i$ , this sum, or the error function is

$$\sum_{i=1}^n |e_i| = \sum_{i=1}^n \left| \sum_{j=1}^n a_{ij}x_j - b_i \right| \quad (4.19)$$

When the  $e_i$ 's vanish simultaneously, that is,  $e_1 = e_2 = \dots = e_n = 0$ , the values of  $x_i$  will be the roots of the system, Eq. (4.2).

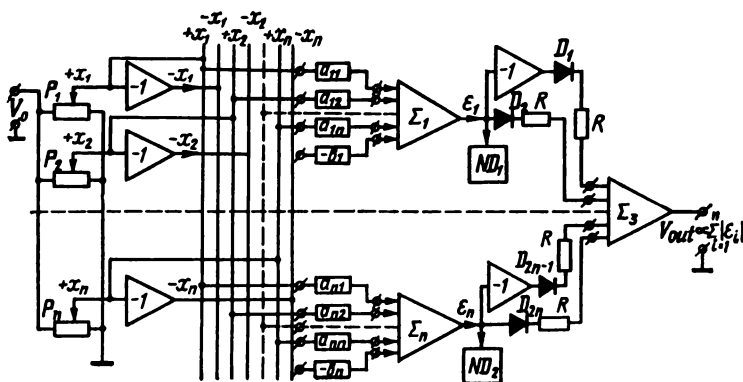


Fig. 4.4

A complete computer set-up to solve Eq. (4.19) by the minimization method appears in Fig. 4.4. Since the summing junction of the output amplifier is at virtual ground potential, a chain of diodes and sign inverters will only pass the positive values of  $e_i$  to its input. The values of the input voltages  $x_i$  are set in various combinations in a sequence of iterative steps, until the output voltage of the summing amplifier,  $\Sigma_3$ , is minimized to zero. This is an indication that

$$\sum_{i=1}^n |e_i| = 0$$

When this occurs, the input voltages will be proportional to the sought roots  $x_i$ . The value of  $V_{out}$  is displayed by a suitable meter. The computer set-up based on the minimization technique markedly simplifies the operator's duty.



# Part Two

## Electronic Digital Computers

### Chapter V

#### Mathematical Basis of Digital Computers

##### 5.1. General

**5.1.1. Structure and basic parameters.** In contrast to analog computers, digital machines operate on quantities represented as numbers. This Part will be concerned with program-controlled digital computers which differ from other digital machines in that (1) they have a large-capacity memory or storage, and data are read into and out of this storage at a speed comparable with that at which the computer itself can operate; (2) problem solving is completely automated.

Input data and intermediate results in a digital computer are represented as finite strings of digits called *words*. In fact, the same method is used to represent not only the numbers involved in a problem, but also any alphabetic information (words, concepts, etc.). For this purpose, the characters of an adopted alphabet are coded with digits. The number of digits in a word adopted for a particular digital computer is called its *word size*, and the word itself is referred to as a *machine word* or a *machine code*.

The process of problem solving on a digital computer is made up of a sequence of arithmetic and logical operations executed to a predetermined program which is based on a problem algorithm. An algorithm is a precise description of the manner in which a particular set of operations (a process of computation) must be carried out in order to solve any problem in a given class.

Numbers are represented as (coded with) electrical signals. In the case of binary representation (to be explained later), 1's and 0's are represented by voltage or current pulses, voltage levels, etc, the presence of a pulse signifying a 1 and its absence, a 0. This method of data representation gives digital computers a marked advantage over analog computers, because the accuracy of computation is only limited by the range of digits in the numbers involved in a problem.

Any present-day electronic digital computer has five major system components or functional sections:

(1) an arithmetic unit, *AU*, to perform arithmetical and logical operations;

(2) a memory or storage unit, *SU*, to store data and instructions for a problem;

(3) an input unit, *INPUT*, to enter information into the computer;

(4) an output unit, *OUTPUT*, to transfer information from the computer;

(5) a control unit, *CU*, to interpret instructions to direct arithmetic and logical operations.

The actual set-up of a particular digital computer and the capabilities of its functional sections depend on applications.

Among the basic parameters describing the quality of various types of digital computer are (1) the speed of operation expressed as the statistical average number of operations performed by a

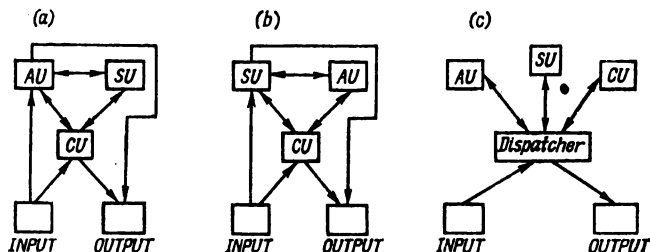


Fig. 5.1

computer per unit time; (2) storage set-up, capacity (the number of words that can be stored at a time) and access time (that is, the time required to write or read an individual data item into or from the storage); (3) set-up and speed of operation of input-output (*I/O*) devices; (4) word size; (5) the basic operations available (known as the computer instruction set); (6) the number of addresses per instruction; (7) reliability in terms of the mean time between failures (*MTBF*); and (8) the average number of operating hours per day, which includes the time a computer has been in the OPERATE (or SOLVE) condition, the DEBUG condition, or has operated trouble-free, etc.

The manner in which the functional units are interconnected and transfer of data is organized within a computer may be traced by reference to its block diagram (Fig. 5.1). In the early models of electronic digital computers, all data read into or out of the computer were arranged to pass through the arithmetic unit (Fig. 5.1a). In later models, the flow of data was through the storage units (Fig. 5.1b). As a result, *I/O* operations could be carried out at the same time with data processing in the arithmetic unit. As a further improvement, the computer structure

was expanded to include several storage units differing in capacity and access time. This provided a still larger scope for the various units to operate simultaneously, thereby enhancing the speed of the computer as a whole. For example, the arithmetic unit could use a small-capacity storage unit having the shortest access time of all and a medium-capacity, medium-access time storage unit, while the I/O devices could employ large-capacity storage units with a long access time. In present-day machines, such as the Soviet-made БЭСМ-6, the flow of data and instructions are organized by a dispatcher (Fig. 5.1c). With a dispatcher, the computer can handle several problems at a time, thereby improving the utilization of computer equipment and computer throughput. The term dispatcher applies to the computer program and to the physical device (or devices) executing it.

**5.1.2. Instructions. Automatic problem solving.** A digital computer solves problems automatically in response to instructions. An instruction is a piece of information arranged as a string of digits of a standard length, called a word, which tells the computer what to do and how. One instruction will cause the computer to execute one operation (which may be arithmetic, transfer, etc.). An algorithm for problem solving arranged as a schedule of instructions is called a program.

The program and the input problem data represented as machine words (in most cases, the words representing instructions and input numbers have the same size) are put into the respective locations of the storage unit. To distinguish one storage location from another each is assigned a number, called an address.

An instruction word consists of two groups of digits. One group represents the operation code (operation No.) and the other, the address of the word on which a given operation is to be carried out (the operand address).

After the requisite instructions and input numbers have been loaded into the storage unit of a digital computer, the process of problem solving reduces to a number of cycles, with one instruction of the program being executed within each cycle. In the case of a three-address instruction set, a standard operating cycle to perform an arithmetical operation will involve the following steps:

- (1) An appropriate instruction is fetched from the storage unit to the control unit.

- (2) The first word located at one of the addresses in the instruction word (say, the first operand address) is transferred from the storage to the arithmetic unit.

- (3) The second word located at the second operand address in the instruction word is transferred from the storage to the arithmetic unit.

(4) The words transferred from the storage are operated upon as prescribed by the operation code in the instruction residing in the control unit.

(5) The result of the operation is transferred to the storage to be located at the third (result) address given in the instruction.

To make problem solving completely automatic, it is important that the instruction next following the first instruction in the program should automatically be transferred to the control unit. This can be done in any one of three ways:

(a) by writing the instruction words of a program in the same order as they are to be executed into storage locations with addresses consecutively incremented by unity. Then the next instruction address will be generated by adding unity to that of the previous instruction;

(b) by using an unconditional transfer instruction (a jump instruction). This instruction specifies the next instruction to be obeyed by the address of the storage location in which it is held. An unconditional jump instruction should be used in cases where some segments of a program must be repeated several times;

(c) by using a conditional transfer instruction (a branch instruction). It enables the computer to decide, at any stage during a computation process, which of two (or more) alternative instructions it will obey next, depending on some criterion to be satisfied by the previous instruction. If this criterion is not met, the next instruction address will be generated according to (a) above. A conditional transfer instruction is used when a computation branches, subject to some conditions.

It is worth while stressing a very important advantage of digital computers — the ability automatically to modify instructions at any stage in a computation. This is a change in an instruction address such that a routine containing the instruction will be repeated on a different number each time. This can be done because both numbers and instructions are represented in coded form by machine words, and both can be operated upon alike. The ability to modify the address part of instructions markedly reduces the number of instructions needed for cyclic computations and, as a consequence, the storage capacity required to store them.

## **5.2. Arithmetic Principles**

**5.2.1. Number systems and their use in electronic digital computers.** A number system is a set of symbols and rules intended to represent numbers. The many number systems may be classed into two broad groups, positional and nonpositional. Although the oldest of all, nonpositional number systems have nearly completely been replaced with systems using positional notation.

In positional notation, a number is represented by a sequence of digits

$$X = x_1 x_2 x_3 \dots x_k \dots x_n$$

in which the value of each digit,  $x_k$ , depends on its position in the sequence.

Any number in a constant-radix positional number system may be represented as

$$X = x_0 \sum_{k=1}^n x_k q^{m-k} = x_0 q^m \left( \sum_{k=1}^n x_k q^{-k} \right)$$

where  $q$  = radix (or base) of the number system

$x_k$  = digit in the  $k$ th position, or  $k$ th digit

$m$  = total quantity of digits in the number

$n$  = exponent of the number

$x_0$  = sign digit

The radix (or base) gives the number of distinguishable symbols, or digits, used in the representation. It also gives the number by which the weight of a position in the representation must be multiplied to obtain the weight of the next more significant position.

In order to represent any  $n$ -digit number (less its sign character) in an electronic digital computer, one needs either  $n$  physical elements each with  $q$  stable states, or  $nq$  elements with only two stable states. Assuming that the physical elements for any number of steady states are identical in terms of performance (reliability, speed, size, cost, etc.), the least amount of equipment for a digital computer in the former case would be needed with a number system having the highest practicable number of steady states,  $q$ . Unfortunately, existing components with more than two stable states (decade counter tubes, stepping relays, etc.) suffer from a number of important drawbacks (large size, low speed, insufficient reliability).

Instead of the usual decimal notation, electronic digital computers mostly use the binary number system. A major advantage of the binary number system is that it can readily be adapted to implementation in a computer and also that Boolean algebra (formal logic) can advantageously be used for the analysis and synthesis of various computation schemes. A serious limitation of the binary representation is the need to convert input data from the decimal to binary notation and output data back from the binary to decimal representation.

In addition to the binary representation, electronic digital computers use the octal and hexadecimal number systems, mainly for program writing. This is because an octal or a hexadecimal number is much shorter than the respective binary number. Also,

binary code can be directly converted into octal or hexadecimal form, because the radices in the octal and hexadecimal notation are integer powers of two.

**5.2.2. Conversion of a number from one number system to another.** In order to convert an integer  $N$  expressed in the radix  $p$  to a number system in the radix  $q$ , the number must consecutively be divided by the radix  $q$  until the last quotient is less than  $q$ . Then in the system to the radix  $q$ , the number  $N$  will be presented as an ordered set of residuals, the most significant digit in  $N$  being the last quotient.

**Example 1.** Convert 95 decimal into the binary system. The decimal number is consecutively divided by 2:

$$\begin{array}{r}
 95 \quad |2 \quad |2 \\
 8 \quad |47 \quad -23 \quad |2 \\
 \underline{15} \quad \underline{4} \quad \underline{2} \quad \underline{11} \quad |2 \\
 -14 \quad -7 \quad -3 \quad -10 \quad -5 \\
 \underline{1} \quad \underline{6} \quad \underline{2} \quad \underline{1} \quad \underline{4} \quad |2 \\
 \quad \quad \underline{1} \quad \underline{1} \quad \quad \underline{1} \quad \underline{2} \quad |2 \\
 \quad \quad \quad \quad \quad \quad \quad \quad \underline{0} \quad \underline{1}
 \end{array}$$



(The arrow shows the direction in which the number should be read.) Thus, the binary equivalent of 95 decimal is 1011111.

In converting a fraction expressed in the radix  $p$  to a number system in the radix  $q$ , the fraction is consecutively multiplied by the radix  $q$ , only the fractional part being multiplied at each step. In the  $q$ -nary system, the fraction will be represented by an ordered sequence of the integer parts of the products, where the most significant digit is the first digit of the product.

**Example 2.** Convert 0.8125 decimal to binary form. The fraction is consecutively multiplied by 2:

$$\begin{array}{r}
 \times 0.8125 \\
 \hline
 2 \\
 \times 1.6250 \\
 \hline
 2 \\
 \times 1.2500 \\
 \hline
 2 \\
 \times 0.5000 \\
 \hline
 2 \\
 \hline
 1.0000
 \end{array}$$

(The arrow indicates the direction in which the number should be read.) Thus, the binary equivalent of 0.8125 decimal is 0.1101.

In the case of a mixed decimal number, it is first separated

into the integer and fractional parts, and each part is then converted as already explained.

**5.2.3. Fixed-point and floating-point representations.** In electronic digital computers, numbers can be represented in any one of two forms: natural (fixed-point) or semilogarithmic (floating-point). In each case, the “+” and “-” signs are coded with a 0 and a 1, respectively, the sign bit being the first bit in the number.

In a fixed-point computer, the decimal point for all numbers (operands and results) remains always in a fixed position so that the exponent,  $m$ , remains unchanged. As a rule, fixed-point computers are arranged to have  $m = 0$ , that is, the numbers the computer has to deal with will lie in the range  $1 > x > -1$  (these are so-called digital numbers). In the computer, a fixed-point number will be represented by a sequence

$$x_0 x_1 x_2 x_3 \dots x_n$$

where  $x_0$  is the sign bit. The code of a positive number

$$x = 0.x_1 x_2 x_3 \dots x_n$$

is identified with the number itself.

Negative numbers may be stored in any one of three ways (only binary numbers will be considered):

(1) In sign and magnitude notation:  
a negative number

$$x = -0.x_1 x_2 x_3 \dots x_n$$

is represented as

$$[x]_1 = 1.x_1 x_2 x_3 \dots x_n$$

In this notation, two representations of zero are used:

$$[+0]_1 = 0.000 \dots 0$$

$$[-0]_1 = 1.000 \dots 0$$

with the “1” on the left of the point indicating a negative number.

(2) In 2's complement notation, a negative number

$$x = -0.x_1 x_2 x_3 \dots x_n$$

is represented as

$$[x]_2 = 1.x'_1 x'_2 x'_3 \dots x'_n$$

where the binary point is followed by the 1's complement of  $|x|$ , if  $x < 0$ , with “1” added in the least significant position. With this representation, only one form is used for zero

$$[0]_2 = 0.000 \dots 0$$

Example:

$$x = -0.11001, \quad [x]_2 = 1.00111$$

(3) In 1's complement notation, a negative number

$$x = -0.x_1x_2x_3 \dots x_k \dots x_n$$

is represented as

$$[x]_3 = 1.\bar{x}_1\bar{x}_2\bar{x}_3 \dots \bar{x}_k \dots \bar{x}_n$$

where

$$[\bar{x}_k] = 1 - x_k$$

Example:

$$x = -0.001101, \quad [x]_3 = 1.110010$$

In 1's complement form, two representations are used for zero:

$$[+0]_3 = 0.000 \dots 0$$

$$[-0]_3 = 1.111 \dots 1$$

Advantages of the fixed-point representation are the simplicity with which arithmetic operations can be carried out and the relatively simple computer set-up required. Among the disadvantages are the difficulty in fitting scale factors in order to fit all numbers into the range of bit positions available, variations in accuracy with word size, and a limited range of numbers.

In the floating-point representation, a number is represented by an exponent, which is a variable quantity and a fractional part (mantissa or magnitude) of the number. Inside a computer this number will be written as

$$y_0y_1y_2y_3 \dots y_p, \quad x_0x_1x_2x_3 \dots x_n$$

where  $y_0y_1y_2y_3 \dots y_p$  is the representation of the exponent,  $y_0$  is the sign bit of the exponent,  $x_0x_1x_2x_3 \dots x_n$  is the mantissa of the number, and  $x_0$  is the sign bit of the number. The sign bit and the most significant bit are usually made unequal, and the magnitude is kept within the range  $\pm 0.5 < x < \pm 1$ . Such numbers are referred to as normalized numbers, and the process of reducing all numbers to this form is called normalization.

The mantissa of a number may be represented in sign-and-magnitude, 2's complement and 1's complement form. The exponent may be represented in 1's complement or 2's complement form, because the only operations performed on the exponents are those of addition and subtraction.

Among the advantages of the floating-point representation are an extended range of numbers, the same percent error for all numbers and, as a consequence, an improved accuracy. Among the disadvantages are a greater count of components needed and a more elaborate computer set-up.



**5.2.4. Binary coded decimal representation.** A group of four binary digits (or bits) known as a tetrad can be used to represent one decimal digit. The result is a mixed representation called the binary-coded decimal (*BCD*) representation. It combines the advantages of the binary and decimal number systems.

Of all *BCD* codes, electronic digital computers have been using three: the 8421 code, the 2421 code, and the excess-3 ( $8421 \pm 3$ ) code.

The 8421 code (Table 5.1) is also known as a *natural-weight code*. In this code, any decimal digit is represented by its binary equivalent. This system is used most in *I/O* devices.

Table 5.1

Decimal numbers	Code weights											
	8	4	2	1	2	4	2	1	8	4	2	1
0	0	0	0	0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	0	1	0	0	1	0	1
3	0	0	1	1	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	1	1	1	0	0	0
6	0	1	1	0	1	1	0	0	1	0	0	1
7	0	1	1	1	1	1	0	1	1	0	1	0
8	1	0	0	0	1	1	1	0	1	0	1	1
9	1	0	0	1	1	1	1	1	1	1	0	0

The 2421 and  $8421 \pm 3$  (excess-3) codes have a distinction in that any decimal digit and its 9's complement are represented by mutually complementing tetrads. This feature provides a simple way to obtain 9's complements by inverting the bits of a tetrad. These systems are used in adders in order to perform the operation of subtraction as the addition of the minuend and a number which is the  $10^n$  complement of the subtrahend (where  $n$  is the number of digits in the subtrahend).

Apart from the 4-bit *BCD* codes, there are also 5-bit, 6-bit and 8-bit *BCD* codes. A very valuable advantage of these systems is their ability to detect coding errors. For example, with the so-called biquinary code in which each decimal digit is represented by five bits containing two 1's and three 0's, the appearance of one or two 1's is an indication of an error.

**5.2.5. Columnwise (bit-by-bit) operations.** Some operations in a digital computer can be performed bit-by-bit (or columnwise), that is on the like digits or columns of different numbers. inde-

pends of adjacent digits or columns. Consider these operations.

(1) Complementing columnwise. In this operation, a sequence  $x_0x_1x_2 \dots x_h \dots x_n$  is transformed into another sequence which is its complement,  $\bar{x}_0\bar{x}_1\bar{x}_2 \dots \bar{x}_h \dots \bar{x}_n$ , where  $\bar{x}_h = 1 - x_h$ .

(2) Taking the sum of two numbers columnwise. This operation consists in taking the modulo 2 sums of the digits in the same positions in different numbers according to the rule:

$$0 \oplus 0 = 0, \quad 0 \oplus 1 = 1, \quad 1 \oplus 0 = 1, \quad 1 \oplus 1 = 0$$

where the  $\oplus$  sign symbolizes the operation of taking a modulo 2 sum.

(3) Columnwise logical addition. This operation consists in taking the sums of the digits in the same positions in two different numbers according to the rule

$$0 \vee 0 = 0, \quad 0 \vee 1 = 1, \quad 1 \vee 0 = 1, \quad 1 \vee 1 = 1$$

where the  $\vee$  sign symbolizes logical addition.

(4) Columnwise logical multiplication. This operation consists in taking the products of the digits in the same positions in two different numbers according to the rule

$$0 \wedge 0 = 0, \quad 0 \wedge 1 = 0, \quad 1 \wedge 0 = 0, \quad 1 \wedge 1 = 1$$

where the  $\wedge$  sign symbolizes logical multiplication.

Complementation columnwise is used to obtain negative numbers in 1's complement form. The modulo 2 sum can be taken to compare two numbers. Bit-by-bit logical addition and multiplication is used to modify instructions and numbers.

**5.2.6. Addition and subtraction of fixed-point numbers with an arbitrary sign.** We shall consider digital numbers, that is, those lying in the range  $-1 < x < 1$ . When adding together numbers in 2's complement representation, the binary carry must be added out of the most significant position of the number into its sign bit. There is no binary carry out of the sign bit. The sum is likewise in 2's complement form, provided  $|x + y| < 1$ . For example,

$$\begin{array}{rcl} x = 0.1001 & + [x]_2 = 0.1001 & \\ y = -0.0101 & + [y]_2 = 1.1011 & \\ \hline & [x + y]_2 = 0.0100 & \end{array}$$
  

$$\begin{array}{rcl} x = -0.1001 & + [x]_2 = 1.0111 & \\ y = -0.0101 & + [y]_2 = 1.1011 & \\ \hline & [x + y]_2 = 1.0010 & \end{array}$$

When adding together numbers in 1's complement representation, an end-around binary carry must be added out of the sign

bit into the least significant bit. The sum is likewise represented in 1's complement form. For example,

$$\begin{array}{rcl}
 x = 0.1101 & + & [x]_3 = 0.1101 \\
 y = -0.0101 & + & [y]_3 = 1.1010 \\
 \hline
 & & 10.0111 \quad [x + y]_3 = 0.1000 \\
 & & \quad \quad \quad \uparrow \\
 & & \quad \quad \quad 1
 \end{array}$$
  

$$\begin{array}{rcl}
 x = -0.1001 & + & [x]_3 = 1.0110 \\
 y = -0.0101 & + & [y]_3 = 1.1010 \\
 \hline
 & & 11.0000 \quad [x + y]_3 = 1.0001 \\
 & & \quad \quad \quad \uparrow \\
 & & \quad \quad \quad 1
 \end{array}$$

The sum of two positive or two negative numbers may be greater than one in absolute value, which is more than the computer's range of numbers can accommodate. Such an overflow situation may cause the loss of significant digits in the result. An indication of overflow is the appearance of a 1 in the sign bit in the sum of positive numbers, or a 0 in the sum of negative numbers. However, overflow detection on this basis is inconvenient because the operator has to remember the sign digits of the operands. The most commonly used overflow detection feature consists in representing the sign of a number with two bits, a "+" sign being coded as 0's and a "-" sign as 1's in both bit positions of the sign. This procedure is known as *modified coding* or *representation*. Accordingly, there is a modified 2's complement representation and a modified 1's complement representation. For modified representations, an overflow indication is the appearance of 01 or 10 in the sign position. For example,

$$\begin{array}{rcl}
 x = 0.1001 & + & [x]_2 = 00.1011 \\
 y = 0.1001 & + & [y]_2 = 00.1001 \\
 \hline
 & & 01.0100
 \end{array}
 \qquad
 \begin{array}{rcl}
 x = -0.1011 & + & [x]_2 = 11.0101 \\
 y = -0.1101 & + & [y]_2 = 11.0011 \\
 \hline
 & & 10.1000
 \end{array}$$

In the first and second examples, two unlike digits, 01 and 10, appear in the sign positions. Thus, an overflow situation has occurred.

Subtraction of fixed-point numbers with an arbitrary sign can be carried out by changing the sign of the subtrahend and adding together the operands.

**5.2.7. Addition and subtraction of floating-point numbers with an arbitrary sign.** Addition or subtraction of floating-point numbers having the same exponent does not differ from the similar operations on fixed-point numbers. If, however, the floating-point numbers differ in exponents, that with the smaller exponent must be altered to have the same exponent as the other before proceeding with addition or subtraction.

The sum or difference of floating-point numbers may be less than 0.5 or greater than 1; that is, the number part of the result may lie outside bounds on the right or left, respectively. When this happens, the result must be shifted, or normalized, left or right, and its exponent altered.

An indication that the result needs shifting or normalization to the right is 01 or 10 appearing in the sign bits of the number. An indication that the result needs normalization to the left (for numbers in 1's complement or 2's complement form, regular or modified) is 0.0 or .1.1 appearing in the least significant bit of the sign and in the most significant digit of the fractional part.

**Example 1.** Add together  $x = 2^3 \times 0.1101$  and  $y = 2^4 \times 0.1010$ . Before the numbers can be added together:

(1) alter them to make their exponents the same:

$$x = 00.01101 \times 2^4$$

$$y = 00.10100 \times 2^4$$

(2) take the sum of the magnitudes

$$\begin{array}{r} 00.01101 \\ + 00.10100 \\ \hline 01.00001 \end{array}$$

The result extends beyond bounds on the left. After the result is normalized by shift to the right and its exponent is incremented by one, we get

$$x + y = 00.100001 \times 2^5$$

**Example 2.** Add together two numbers in 2's complement representation:

Given:

$$\begin{array}{l} x = 0.1101 \times 2^4; \quad y = -0.1010 \times 2^4 \\ [x]_2 = 00.1101 \times 2^4 \\ [y]_2 = 11.0110 \times 2^4 \\ \hline [x + y]_2 = 00.0011 \times 2^4 \end{array}$$

The result extends beyond bounds on the right. After the result is shifted to the left and the exponent altered, we get

$$[x + y]_2 = 0.1100 \times 2^2$$

**5.2.8. Taking the product of fixed- and floating-point numbers.** In a digital computer, the multiplication of binary numbers is accomplished by the combined operations of addition and shifting. Taking the product of two numbers is controlled by the digits of the multiplier.

If a bit of the multiplier is 1, the multiplicand is shifted as many places as may be necessary and added to the partial pro-

duct. If a bit of the multiplier is 0, the multiplicand is not added. In most cases, electronic digital computers perform multiplication on numbers in the sign-and-magnitude representation. The choice of procedures for multiplication is summarized in Table 5.2.

Table 5.2

Start of multiplication	Proce- dure	Direction of shift		
		for multi- plicand	for multi- plier	for sum of partial products
From least significant digits of multiplier	1	stationary	right	right
	2	left	right	stationary
From most significant digits of multiplier	3	stationary	left	left
	4	right	left	stationary

Consider procedure 1, often employed in electronic digital computers. Let the multiplicand be  $x = 0.10100$  and the multiplier,  $y = 0.11001$ .

Adder	Multiplier	
$+ \begin{array}{r} 0.00000 \\ 0.10100 \end{array}$	11001	Send $x$ to adder
$\begin{array}{r} 0.10100 \\ 0.01010 \\ 0.001010 \\ + 0.0001010 \\ 0.10100 \end{array}$	$\begin{array}{r} 11001 \\ 01100 \\ 0110 \\ 011 \end{array}$	$\begin{array}{l} \text{Shift right one place} \\ \text{Shift right one place} \\ \text{Shift right one place} \\ \text{Send } x \text{ to adder} \end{array}$
$\begin{array}{r} 0.10110100 \\ + 0.01011010 \\ 0.10100 \end{array}$	$\begin{array}{r} 11 \\ 01 \end{array}$	$\begin{array}{l} \text{Shift right one place} \\ \text{Send } x \text{ to adder} \end{array}$
$\begin{array}{r} 0.111110100 \\ 0.011111010 \end{array}$	$\begin{array}{r} 1 \\ 0 \end{array}$	$\begin{array}{l} \text{Shift right one place} \\ \text{Product} \end{array}$

An advantage of the multiplication procedures in which the sum of partial products remains stationary is that the multiplicand

and multiplier can be shifted and the operation of addition performed during the same clock time.

Where the product is to have  $m$ , and not  $2m$  digits, it is advantageous to use procedure 4 (see Table 5.2); it involves a smaller count of equipment and a shorter multiplication time. This is because instead of  $2m$  digits the adder and the multiplier and register need to have  $m + n$  digits. Here,  $n$  is the number of additional digits that must be provided to give the desired accuracy and to accommodate the multiplicand and multiplier bits (for  $m = 15$  and an error of not over one bit,  $n = 4$ ). The sign of the product is obtained by taking the modulo 2 sum of the sign bits of the operands.

In floating-point multiplication, in addition to taking the product of the numbers and the sum of their sign bits, one has to find the exponent of the product and normalize the result. The exponent of the product is found by adding together the exponents of the operands.

**5.2.9. Division of fixed- and floating-point numbers.** In electronic digital computers, division is done by the combined operations of subtraction, addition and shifting. The sign of the quotient is obtained by taking the modulo 2 sum of the sign bits of the dividend and divisor. At least two ways exist for performing it, respectively known as restoring division and nonrestoring division.

In restoring division of numbers stored as sign and magnitude, the algorithm is as follows:

(a) the dividend must be smaller than the divisor (for fixed-point computers). Shift the dividend one place left and subtract the divisor from the shifted dividend;

(b) if the difference is greater than zero, the first digit of the quotient will be a 1; if the difference is less than zero, the first digit of the quotient will be a 0. After subtraction, in the former case, the remainder is shifted left one place and the procedure is repeated, starting with step (a). In the latter case, the divisor is added to the remainder before shifting. With restoring division, two operations of subtraction and addition are required to obtain one digit in the quotient.

In nonrestoring division, the algorithm is as follows:

(a) the initial steps are the same as in (a) and (b) for restoring division;

(b) the difference is shifted left one place. If the remainder is positive, subtract the divisor from the shifted remainder. If the remainder is negative, add the divisor to the shifted remainder. With nonrestoring division, only one operation of addition or subtraction is needed to obtain one digit of the quotient.

**Example.** Divide 0.10011 by 0.11010.

The dividend is  $x = 0.10011$ , the divisor is  $y = 0.11010$

$$[-y]_2 = 11.00110$$

Adder	Quotient register	
$\begin{array}{r} 00.10011 \\ 01.00110 \\ + 11.00110 \end{array}$	$\begin{array}{r} 00000 \\ 0000- \end{array}$	Shift left one place Send $[-y]_2$ to adder
$\begin{array}{r} 00.01100 \\ 00.11000 \\ + 11.00110 \end{array}$	$\begin{array}{r} 00001 \\ 0001- \end{array}$	Shift left one place Send $[-y]_2$ to adder
$\begin{array}{r} 11.11110 \\ 11.11100 \\ + 00.11010 \end{array}$	$\begin{array}{r} 00010 \\ 0010- \end{array}$	Shift left one place Send $y$ to adder
$\begin{array}{r} 00.10110 \\ + 01.01100 \\ 11.00110 \end{array}$	$\begin{array}{r} 00101 \\ 0101- \end{array}$	Shift left one place Send $[-y]_2$ to adder
$\begin{array}{r} 00.10010 \\ + 01.00100 \\ 11.00110 \end{array}$	$\begin{array}{r} 01011 \\ 1011- \end{array}$	Shift left one place Send $[-y]_2$ to adder
00.01010	10111	

The result is  $x/y = 0.10111$ , the remainder is  $0.01010 \times 2^{-5}$ .

For floating-point numbers, in addition to dividing their magnitudes and finding the sign of the quotient, division involves finding the exponent and normalization of the quotient. The exponent of the quotient is found in the exponent adder by subtracting the exponent of the divisor from that of the dividend. The procedure starts with finding the units digit prior to shifting by the rules defined above. If the units digit is a 1, the quotient must be normalized by shifting it right after division.

**5.2.10. Round-off.** Shifting a number right may yield a result too long for the computer to store, and the number may have then to be truncated, which involves the loss of the least significant digit. This truncation error is less than a 1 in the least significant digit place of the result. To minimize this error, it is the usual practice to apply a round-off correction. For this purpose, a digit place is added to the right of the last place to be retained in the adder. After a number has been shifted right, a 1 is added to or subtracted from the extra digit place, according to the number representation. For a number in 2's complement form, a 1 is added always; for a number in 1's complement form, a 1 is added if the sign bit of the number in the adder contains a 0, and it is subtracted if the sign bit contains a 1. This round-off correction reduces the truncation error to one-half of a 1 in the last significant digit of the result.

### 5.3. Switching Circuits and Boolean Algebra

**5.3.1. Switching circuits defined.** All digital computer components are examples of *digital* or *switching circuits* with a varying degree of sophistication. In the analysis and synthesis of such devices, use is made of the theory of finite-state machines or automata. In this book, only the most important aspects of this theory essential to a proper understanding of operation and design of switching circuits for digital computers will be given in Chapters 5 and 6.

Before going any further, some concepts and definitions must be introduced. A switching (or digital) circuit is defined as a device intended to process digital information, ideally having at its input and at its output terminals signals that may each have one of just two possible values.

Switching circuits may be grouped into two broad classes, *sequential circuits* and *combinational circuits*. The former have more than one internal state, and the latter only just one internal state. Sometimes they are called switching circuits with and without memory, respectively. The latter are also called 'primitive circuits'.

Practical switching circuits are finite-state. This indicates that they have a finite set of input and output signals, a finite number of input and output channels, and a finite number of internal states.

Switching circuits operate at discrete time intervals known as *digit times*,  $T$ . At each digit time, precisely one signal (digit) is applied to the input and precisely one signal is available at the output of a switching circuit, associated with a particular internal state. Depending on how the internal state is allowed to



change with time, switching circuits may be divided into *synchronous* and *asynchronous*. Synchronous switching circuits are allowed to change state in synchronism with clock pulses from an internal or external clock generator which generates these signals at  $T = \text{constant}$ . For asynchronous switching circuits  $T \neq \text{constant}$ , that is, each individual element proceeds at its own speed upon arrival of an input signal, and a change of state occurs while the state of the input remains unchanged. For idealized switching circuits, that is, ones in which transients are ignored, the difference in actual values of  $T$  is immaterial for proper functioning. Therefore, in describing the behaviour of switching circuits one introduces an abstract switching time which can take on nonzero integer values,  $t = 0, 1, 2, \dots$

The specification of a switching circuit calls for knowledge of three alphabets: input alphabet,  $X = [x_1, x_2, \dots, x_m]$ ; output alphabet,  $Y = [y_1, y_2, \dots, y_n]$ ; and state alphabet,  $A = [a_0, a_1, a_2, \dots, a_r]$ . Then, the law governing the behaviour of a switching circuit may be defined as

$$a(t+1) = f[a(t), x(t)] \quad (5.1)$$

$$y(t) = \phi[a(t), x(t)] \quad (5.2)$$

$$a(0) = a_0 \quad (5.3)$$

where  $f$  = transition function of the switching circuit

$\phi$  = output function of the switching circuit

$a_0$  = initial state of the switching circuit

$a(t)$  = state at time  $t$

$x(t)$  = input at time  $t$

$y(t)$  = output at time  $t$

Switching circuits whose behaviour is governed by Eqs. (5.1), (5.2) and (5.3) fall under the category of *Mealy automata*. In contrast, there are switching circuits whose output signals are solely determined by the internal state of the circuit and are independent of the magnitudes of input signals. They fall under the category of *Moore automata*.

Most commonly, switching circuits are specified by describing their terminal action with the aid of transition (or flow) tables and state tables (Tables 5.3 and 5.4). These tables represent the mapping of a set out of the input alphabet into a set out of the output alphabet. That is, for any input word out of the input alphabet there will be a certain definite word out of the output alphabet. Each time an input word is to be applied to a switching circuit it is reset to the initial state. As an example, in the case of the switching circuit specified in Tables 5.3 and 5.4, an arbit-

Table 5.3

$x \backslash a$	$a_0$	$a_1$	$a_2$	$a_3$
$x_1$	$a_2$	$a_2$	$a_3$	$a_3$
$x_2$	$a_1$	$a_3$	$a_2$	$a_0$

Table 5.4

$x \backslash a$	$a_0$	$a_1$	$a_2$	$a_3$
$x_1$	$y_1$	$y_2$	$y_2$	$y_2$
$x_2$	$y_1$	$y_1$	$y_3$	$y_1$

rary set of input signals,  $x_1x_1x_2x_1x_2x_1x_2x_2x_1$ , will be mapped into the following sets of states and output signals:

$$a_0a_2a_3a_0a_2a_3a_0a_1$$

$$y_1y_2y_1y_1y_3y_2y_1y_2$$

If two digital (switching) circuits having the same input and output alphabets produce the same mapping of a set out of the input alphabet into a set out of the output alphabet, the circuits are said to be equivalent.

**5.3.2. Boolean functions. Basic concepts and definitions.** In most cases, electronic digital computers use binary alphabet as it greatly simplifies the development of block and flow diagrams. Since information is presented in binary (bivalued) form, it has been found convenient to employ Boolean algebra in the analysis and synthesis of the fundamental logical circuits in digital computers.

What are then a Boolean (bivalued) variable and a Boolean (bivalued) function? *Boolean*, or *bivalued functions*  $x_1, x_2, \dots, x_n$  are variables which only take two values called "true" and "false" or, more simply, "1" and "0". The values of a Boolean variable form a set symbolized as  $x_1, x_2, \dots, x_i, \dots, x_n$  (where  $x_i$  may be a 0 or a 1).

A Boolean or bivalued function of two binary variables is a function which takes only two values, "1" and "0". The domain of a Boolean function is finite, because the arguments of the function are bivalued. The total number of sets of binary arguments for which a Boolean function is defined is  $2^n$ .

Any Boolean function may be specified by giving its truth table (Table 5.5) which gives the binary value of the function (on the right) for each combination of binary variables (on the left). The term 'truth table' reflects the fact that if 1's are identified with true sentences and 0's with false ones, the table will prove or disprove the validity of a complex sentence depending on whether the constituent sentences are true or false. It can be easily seen from the construction of the truth table that  $n$  independent va-

riables lead to  $2^{2^n}$  Boolean functions. By induction, this may be proved as follows. A single set of independent variables leads to two Boolean functions, two sets to  $2^2$ ,  $k$  sets to  $2^k$ , and finally  $2^n$  sets,  $2^{2^n}$  Boolean functions.

Table 5.5

$x_1 x_2 x_3 \dots x_{n-1} x_n$	$f(x_1, x_2, x_3, \dots, x_{n-1}, x_n)$
000...00	$f_0(x_1, x_2, x_3, \dots, x_{n-1}, x_n)$
000...01	$f_1(x_1, x_2, x_3, \dots, x_{n-1}, x_n)$
000...10	$f_2(x_1, x_2, x_3, \dots, x_{n-1}, x_n)$
...	...
011...11	$f_{2^{n-2}}(x_1, x_2, x_3, \dots, x_{n-1}, x_n)$
111...11	$f_{2^n-1}(x_1, x_2, x_3, \dots, x_{n-1}, x_n)$

In the analysis and synthesis of circuits for electronic digital computers, wide use is made of Boolean functions of one and two variables. One variable leads to only four distinct Boolean functions. Their truth tables are combined in Table 5.6.

Table 5.6

$x$	$g_1$	$g_2$	$g_3$	$g_4$
0	0	0	1	1
1	0	1	0	1

As is seen, the functions  $g_1$  and  $g_4$  are constants 0 and 1, respectively, and the function  $g_2$  repeats the value of the variable  $x$ , that is,  $g_2 = x$ . The function  $g_3$  is commonly called *not*, *negation* or *complementation*, and is symbolized as  $\bar{x}$ , that is,  $g_3 = \bar{x}$ .

Two variables lead to a total of 16 Boolean functions (Table 5.7).

Table 5.7

$x_1$	$x_2$	$f_0$	$f_1$	$f_2$	$f_3$	$f_4$	$f_5$	$f_6$	$f_7$	$f_8$	$f_9$	$f_{10}$	$f_{11}$	$f_{12}$	$f_{13}$	$f_{14}$	$f_{15}$
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

These sixteen functions include the so-called degenerate functions:

- $f_0(x_1, x_2) = 0$ , trivial (no dependence on the inputs)
- $f_{15}(x_1, x_2) = 1$ , trivial (no dependence on the inputs)
- $f_3(x_1, x_2) = x_1$ , special case (depends on one input)
- $f_5(x_1, x_2) = x_2$ , special case (depends on one input)
- $f_{12}(x_1, x_2) = \bar{x}_1$ , complement of  $x_1$
- $f_{10}(x_1, x_2) = \bar{x}_2$ , complement of  $x_2$

The remaining ten functions of two variables, along with their symbols and names, are presented in Table 5.8. Using functions of one or two binary variables, called *elementary logical functions*, and the superposition principle, one can readily derive any other Boolean function.

Table 5.8

Function	Name	Algebraic notation	To be read as
$f_1(x_1, x_2)$	Conjunction	$x_1 x_2$	$x_1$ and $x_2$
$f_7(x_1, x_2)$	Disjunction	$x_1 \vee x_2$	$x_1$ or $x_2$ (or both)
$f_6(x_1, x_2)$	Exclusive disjunction	$x_1 \bar{x}_2 \vee \bar{x}_1 x_2$	$x_1$ or $x_2$ but not both
$f_8(x_1, x_2)$	Pierce function (dual stroke)	$\bar{x}_1 \bar{x}_2$	Neither $x_1$ nor $x_2$
$f_9(x_1, x_2)$	Equivalence	$\bar{x}_1 \bar{x}_2 \vee x_1 x_2$	$x_1$ if and only if $x_2$
$f_{11}(x_1, x_2)$	Implication	$x_1 \vee \bar{x}_2$	if $x_2$ then $x_1$
$f_{14}(x_1, x_2)$	Sheffer function (stroke)	$\bar{x}_1 \vee \bar{x}_2$	Not both $x_1$ and $x_2$
$f_2(x_1, x_2)$	Inhibit on $x_2$	$x_1 \bar{x}_2$	Not if $x_1$ then $x_2$
$f_4(x_1, x_2)$	Inhibit on $x_1$	$\bar{x}_1 x_2$	Not if $x_2$ then $x_1$
$f_{13}(x_1, x_2)$	Implication	$x_2 \vee \bar{x}_1$	If $x_1$ then $x_2$

Treating Boolean functions of one or two variables as operations on the set of all Boolean functions, we can construct different other Boolean algebras. In our further exposition, we shall be interested in a two-element Boolean algebra with three binary operations of negation, conjunction (multiplication) and disjunction (addition).

**5.3.3. Boolean expressions. Basic laws.** Boolean functions can conveniently be described algebraically by Boolean expressions. This involves the use of certain symbols. In addition to the symbols of the three basic binary operations of negation, conjunction and disjunction, there are the lower-case letters from the end of the alphabet, like  $x$ ,  $y$  and  $z$  (together with appropriate subscripts) to designate variables, the constants 0 and 1, and a pair of parentheses, ( ).

A Boolean expression, defined below, is a finite string of the symbols listed above and meeting the definition of a Boolean expression. Boolean expressions are:

- (a) Boolean variables  $x, y, z, \dots$
- (b) The constants 0 and 1.
- (c) If  $A$  and  $B$  are expressions, then so are  $(A \vee B)$ ,  $(A \wedge B)$  and  $\bar{A}$ .

For example,  $((x_1 \wedge x_2) \vee x_3) \wedge (x_4 \wedge x_5)$  is an expression, but  $(A \wedge B, \vee A)$  is not.

Extra parentheses can often be dropped by the convention that a negation is weaker than a conjunction and a conjunction is weaker than a disjunction.

Boolean expressions are equal (equivalent), if the functions assigned to them are equal, that is, if the functions take on the same values for every combination of assignments or values for the variables.

There are certain axioms, laws and theorems by which identical transformations can be carried out on Boolean expressions. Some of them are stated below:

- (1) Axiom of double-negation:

$$\bar{\bar{x}} = x \quad (5.4)$$

- (2) Axiom of commutativity of disjunction and conjunction

$$\begin{aligned} x_1 \vee x_2 &= x_2 \vee x_1 \\ x_1 x_2 &= x_2 x_1 \end{aligned} \quad (5.5)$$

- (3) Axiom of associativity of disjunction and conjunction

$$\begin{aligned} x_1 \vee (x_2 \vee x_3) &= (x_1 \vee x_2) \vee x_3 \\ x_1 (x_2 x_3) &= (x_1 x_2) x_3 \end{aligned} \quad (5.6)$$

- (4) Axiom of distributivity

$$\begin{aligned} x_1 (x_2 \vee x_3) &= x_1 x_2 \vee x_1 x_3 \\ x_1 \vee x_2 x_3 &= (x_1 \vee x_2) (x_1 \vee x_3) \end{aligned} \quad (5.7)$$

- (5) De Morgan's laws or theorems

$$\begin{aligned} \overline{x_1 \vee x_2} &= \bar{x}_1 \bar{x}_2 \\ \overline{x_1 x_2} &= \bar{x}_1 \vee \bar{x}_2 \end{aligned} \quad (5.8)$$

- (6) Axiom for operations on the constants 0 and 1:

$$\begin{aligned} \bar{0} &= 1, \quad \bar{1} = 0 \\ x \cdot 1 &= x, \quad x \cdot 0 = 0 \\ x \vee 0 &= x, \quad x \vee 1 = 1 \end{aligned} \quad (5.9)$$

(7) Axiom for operations on a variable and its complement

$$\begin{aligned}x \vee \bar{x} &= 1 \\ x\bar{x} &= 0\end{aligned}\tag{5.10}$$

The above axioms and laws can be proved by consecutively substituting all assignments or values for the variables in the relations.

The following important relationships or tautologies can be deduced from the above axioms and laws.

(1) Absorption

$$\begin{aligned}x_1 \vee x_1x_2 &= x_1 \\ x_1(x_1 \vee x_2) &= x_1\end{aligned}\tag{5.11}$$

(2) Idempotence of disjunction and conjunction

$$\begin{aligned}x \vee x &= x \\ xx &= x\end{aligned}\tag{5.12}$$

(3) From de Morgan's laws and using mathematical induction, any Boolean expression formed with the operations of conjunction, disjunction and negation can be negated by replacing the arguments in the original expression with their negations and by exchanging the symbols of conjunction and disjunction.

(4) By the second axiom of distributivity,

$$x_1 \vee \bar{x}_1x_2 = x_1 \vee x_2\tag{5.13}$$

**5.3.4. Forms of Boolean functions.** In Boolean algebra it is proved that any Boolean function, except the function  $f = 0$ , can be expressed in terms of conjunction, disjunction and negation as follows

$$z = f(x_1, x_2, \dots, x_k, \dots, x_n) = \vee x_1^{\sigma_1}x_2^{\sigma_2} \dots x_k^{\sigma_k} \dots x_n^{\sigma_n}\tag{5.14}$$

where  $x_k^{\sigma_k}$  is the common designation for the argument  $x_k$  and its negation  $\bar{x}_k$ , such that

$$x_k^{\sigma_k} = \begin{cases} x_k & \text{for } \sigma_k = 1 \\ \bar{x}_k & \text{for } \sigma_k = 0 \end{cases}$$

In Eq. (5.14), logical addition is performed for those sets  $\sigma_1, \sigma_2, \dots, \sigma_n$  for which  $f(\sigma_1, \sigma_2, \dots, \sigma_n) = 1$ . This form is called the (expanded) *disjunctive normal form*. The terms of the expanded disjunctive normal form are called basic conjunctions or min-terms.

A Boolean function specified in tabular form may be expressed in its expanded disjunctive normal form as follows: the orderings of the variables for which the function is true, that is, equal to

unity, are selected from the table; for each set of variables a conjunction is written up, such that  $x_1^{\sigma_1} x_2^{\sigma_2} \dots x_n^{\sigma_n}$ ; then the conjunctions are linked together with the disjunction connective to give the expanded disjunctive normal form of the function.

**Example.** Expand the function specified by Table 5.9 into its disjunctive normal form.

Table 5.9

$x_1$	$x_2$	$x_3$	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Recalling the rule stated above, we obtain

$$z = \bar{x}_1 x_2 x_3 \vee x_1 \bar{x}_2 x_3 \vee x_1 x_2 \bar{x}_3 \vee \bar{x}_1 \bar{x}_2 x_3$$

As an alternative, a Boolean function may be expressed in the expanded conjunctive form. In this case, one forms a conjunction of those basic disjunctions of its variables for which the function is true. For the function  $f(x_1, x_2, x_3)$  discussed above, its conjunctive normal form expansion will be

$$\begin{aligned} z &= f(x_1, x_2, x_3) \\ &= (x_1 \vee x_2 \vee x_3)(x_1 \vee \bar{x}_2 \vee x_3)(\bar{x}_1 \vee x_2 \vee x_3)(\bar{x}_1 \vee \bar{x}_2 \vee x_3) \end{aligned}$$

In going from the expanded disjunctive normal form of a Boolean function to its expanded conjunctive form, one takes the logical sum of the basic disjunctions which are not in the disjunctive normal form, that is, the negation of the function, after which the negation of the sum is taken. For example,

$$\begin{aligned} z &= f(x_1, x_2) = \bar{x}_1 x_2 \vee x_1 \bar{x}_2, \quad \bar{z} = x_1 x_2 \vee \bar{x}_1 \bar{x}_2 \\ z &= \bar{\bar{z}} = (\bar{x}_1 \vee \bar{x}_2)(x_1 \vee x_2) \end{aligned}$$

A similar procedure applies to going from the conjunctive to the disjunctive normal form of a Boolean function.

**5.3.5. Functional completeness of a set of Boolean functions.** A Boolean function or a set of Boolean functions  $\varphi_1, \varphi_2, \dots, \varphi_m$ , is functionally complete if all other functions can be formed from them by superposition of  $\varphi_1, \varphi_2, \dots, \varphi_m$ .

For example, the set of three Boolean functions,

$$\varphi_1 = \bar{x}$$

$$\varphi_2 = x_1 x_2$$

$$\varphi_3 = x_1 \vee x_2$$

is functionally complete, because any Boolean function can be expressed in terms of conjunction, disjunction and negation, Eq. (5.14).

Examples of the sets of elementary Boolean functions that are functionally complete are:

$$\varphi_1 = \bar{x}, \quad \varphi_2 = x_1, \quad x_2$$

$$\varphi_1 = \bar{x}_1, \quad \varphi_2 = x_1 \vee x_2$$

$$\varphi = \bar{x}_1 \vee \bar{x}_2$$

$$\varphi = \bar{x}_1 \bar{x}_2$$

$$\varphi_1 = x_1 \oplus x_2, \quad \varphi_2 = x_1 x_2, \quad \varphi_3 = 1$$

$$\varphi_1 = x_1 \bar{x}_2, \quad \varphi_2 = 1$$

The functional completeness of the above sets of elementary functions can be proved by reducing them to a functionally complete set of three elementary functions  $\bar{x}$ ,  $x_1 x_2$ , and  $x_1 \vee x_2$ . As an example, there is no disjunction in the first set. It may readily be obtained from

$$x_1 \vee x_2 = \overline{\bar{x}_1 \bar{x}_2}$$

There is no conjunction in the second set; it may be obtained from

$$x_1 x_2 = \overline{\bar{x}_1 \vee \bar{x}_2}$$

Thus, the first and second sets are functionally complete because they are reducible to a complete set. The functional completeness of the other sets of functions can be proved in a similar manner.

**5.3.6. Minimization of Boolean functions.** *Minimization* is a process by which logical functions are reduced to a form permitting a simpler implementation with a smaller count of physical devices. Part of the overall process is the minimization of Boolean (or switching) functions. Essentially, one seeks to reduce them to a form involving the use of the least possible number of elements (literals) and the least number of operations on them.

To simplify the understanding of minimization, it is necessary to have available some definitions. A conjunction of the form  $p = x_1^{\sigma_1} x_2^{\sigma_2} \dots x_k^{\sigma_k}$  is called minimal if the number of terms is smaller than some set of variables,  $n$ , with any variable  $x_i^{\sigma_i}$  appearing in the conjunction at most once. The number of terms in a minimal conjunction defines its rank.



An *implicant* of a Boolean function,  $f(x_1, x_2, \dots, x_n)$ , is a Boolean function,  $f_1(x_1, x_2, \dots, x_n)$ , which, being equal to 1 for any ordering of the variables, is also equal to the value of the function  $f$  for this ordering. A *prime implicant* of a Boolean function,  $f(x_1, x_2, \dots, x_n)$ , is any minimal product,  $P = x_1^{\sigma_1} x_2^{\sigma_2} \dots x_k^{\sigma_k}$  which is an implicant of the function  $f$  and no part of which is contained in the function  $f$ . Thus, prime implicants are minimal conjunctions of the lowest rank, appearing in a given Boolean function.

A *contracted disjunctive normal form* of a Boolean function is a disjunction of its all prime implicants. Although the contracted normal form of a Boolean function contains fewer terms than the expanded normal form of the same function, it can, in most cases, be simplified still more due to the absorption of some prime implicants by the disjunction of other prime implicants. If no more implicants can be removed from a disjunction of the prime implicants representing a given Boolean function, this disjunction is called a *dead-end disjunctive normal form*. It should be noted that some Boolean functions may have several dead-end disjunctive normal forms. The dead-end disjunctive normal form of a Boolean function containing the least number of elements is called the *minimal disjunctive normal form*.

Thus, the overall task of minimizing Boolean functions may be accomplished as follows. To begin with, a contracted disjunctive normal form, that is all prime implicants are found for the given function. As the next step, the dead-end disjunctive normal forms of the function are found. Finally, the minimal disjunctive normal form is chosen from among them.

Most often, Boolean functions are simplified to their contracted disjunctive normal forms by the *Quine method*. For the method to be applicable, the given Boolean function must be expressed in its expanded disjunctive normal form. If the given function appears in an arbitrary disjunctive normal form, it is expanded by multiplying some of its terms by an expression of the form  $x \vee \bar{x} = 1$ . Then the Quine method reduces to consecutively applying to pairs of basic conjunctions (the terms of the disjunction) the operations of amalgamation and absorption. The operation of amalgamation is based on the identity

$$x_1 x_2 \vee x_1 \bar{x}_2 = x_1 (x_2 \vee \bar{x}_2) = x_1$$

The operation of absorption is based on the identity

$$x_1 \vee x_1 x_2 = x_1 (1 \vee x_2) = x_1$$

As the first step, all the possible operations of amalgamation are applied to the basic conjunctions of rank  $n$  (where  $n$  is the

number of variables) in the expanded disjunctive normal form of the original function. As a result, conjunctions of rank  $(n - 1)$  are obtained. Now the operations of absorption are applied to the conjunctions of rank  $(n - 1)$ , after which all possible operations of amalgamation are performed on the conjunctions of rank  $(n - 1)$ . Then the operations of absorption are applied to the conjunctions of rank  $(n - 2)$ , and this is followed by operations of amalgamation again, etc.

**Example 1.** Reduce the Boolean function

$$f(x_1, x_2, x_3, x_4) = \bar{x}_1 x_4 \vee x_1 x_3 \vee \bar{x}_3 \bar{x}_4 \vee \bar{x}_1 x_2 \bar{x}_3$$

to its contracted disjunctive normal form.

To begin with, the original Boolean function is expressed in an expanded disjunctive normal form:

$$\begin{aligned} (x_1, x_2, x_3, x_4) = & \bar{x}_1 x_2 \bar{x}_3 x_4 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 x_4 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 \vee \bar{x}_1 x_2 \bar{x}_3 \bar{x}_4 \\ & \vee \bar{x}_1 \bar{x}_2 x_3 x_4 \vee \bar{x}_1 x_2 x_3 x_4 \vee x_1 \bar{x}_2 x_3 \bar{x}_4 \vee x_1 \bar{x}_2 x_3 x_4 \\ & \vee x_1 x_2 x_3 \bar{x}_4 \vee x_1 x_2 x_3 x_4 \vee x_1 x_2 \bar{x}_3 \bar{x}_4 \vee x_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 \end{aligned}$$

where all the basic conjunctions are numbered consecutively.

Now, all the possible operations of amalgamation will be applied to the basic conjunctions in the following sequence: (a) the first conjunction will be amalgamated with all the remaining ones; (b) the second conjunction with all the remaining ones except the first; (c) the third conjunction with all the remaining ones except the first and second, etc. At the end of all amalgamations and absorptions, we shall get the given Boolean function in the following disjunctive normal form:

$$\begin{aligned} f(x_1, x_2, x_3, x_4) = & x_1 x_3 x_4 \vee x_1 x_2 x_3 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 \vee \bar{x}_1 \bar{x}_3 \bar{x}_4 \\ & \vee \bar{x}_2 \bar{x}_3 \bar{x}_4 \vee x_2 x_3 x_4 \vee x_1 x_3 x_4 \vee \bar{x}_2 x_3 x_4 \vee x_2 \bar{x}_3 \bar{x}_4 \\ & \vee x_1 x_2 x_3 \vee \bar{x}_1 x_3 x_4 \vee x_1 x_2 \bar{x}_4 \vee \bar{x}_1 \bar{x}_3 x_4 \vee \bar{x}_1 x_2 x_4 \\ & \vee \bar{x}_1 \bar{x}_3 \bar{x}_4 \vee x_1 \bar{x}_2 \bar{x}_4 \vee x_1 x_3 \bar{x}_4 \end{aligned}$$

where all terms are numbered consecutively. After all the operations of rank 3 performed in the same sequence as before and also the operations of absorption, we get

$$f(x_1, x_2, x_3, x_4) = \bar{x}_1 \bar{x}_3 \vee \bar{x}_1 x_4 \vee x_1 x_3 \vee x_1 \bar{x}_4 \vee x_3 x_4 \vee \bar{x}_3 \bar{x}_4$$

Dead-end and minimal disjunctive normal forms can be derived from contracted disjunctive normal form, using tables of prime implicants. In a table of prime implicants, the rows are defined

by the prime implicants and the columns, by the minterms of the function. If an implicant is included in a minterm, a check mark (say, a cross) is entered at the intersection of the implicant row and the minterm column.

A table of prime implicants (Table 5.10) compiled according to the above rules for the contracted disjunctive normal form of the Boolean function in the above example is given below.

The minimal disjunctive normal form of a given function will be a system of a minimal number of rows the implicants of which cover between them all the columns of the table.

For the case on hand, two minimal disjunctive normal form expressions are possible:

$$f(x_1, x_2, x_3, x_4) = \bar{x}_1\bar{x}_3 \vee x_1\bar{x}_4 \vee x_3x_4 \quad (a)$$

$$f(x_1, x_2, x_3, x_4) = \bar{x}_1x_4 \vee x_1x_3 \vee \bar{x}_3\bar{x}_4 \quad (b)$$

Expression (a) corresponds to the combination of rows 1, 4 and 5; expression (b), to the combination of rows 2, 3 and 6.

Table 5.10

Nos.	Prime Implicant	Minterms											
		$\bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4$	$\bar{x}_1\bar{x}_2\bar{x}_3x_4$	$\bar{x}_1\bar{x}_2x_3\bar{x}_4$	$\bar{x}_1\bar{x}_2x_3x_4$	$\bar{x}_1x_2\bar{x}_3\bar{x}_4$	$\bar{x}_1x_2\bar{x}_3x_4$	$x_1\bar{x}_2\bar{x}_3\bar{x}_4$	$x_1\bar{x}_2\bar{x}_3x_4$	$x_1\bar{x}_2x_3\bar{x}_4$	$x_1\bar{x}_2x_3x_4$	$x_1x_2\bar{x}_3\bar{x}_4$	$x_1x_2\bar{x}_3x_4$
1	$x_1\bar{x}_4$							×	×				
2	$\bar{x}_1x_4$		×	×		×	×				×	×	
3	$x_1x_3$								×	×		×	×
4	$\bar{x}_1\bar{x}_3$	×	×		×	×							
5	$x_3x_4$			×			×			×			
6	$\bar{x}_3\bar{x}_4$	×			×			×			×		

Another convenient representation for simplifying Boolean functions has been devised by Veitch, and known as the *Veitch diagram*. The Veitch diagram has a unit square for each minterm of the Boolean variables, each square corresponding to a row entry in a truth table. All squares which correspond to the basic conjunctions of the original function in the expanded disjunctive normal form have a value of 1, and all the remaining squares have a value of 0. In the Veitch diagram, any pair of adjoining basic conjunctions appear in adjoining squares vertically and horizontally. In the Veitch diagram for three variables (Fig. 5.2), the squares in the first and last columns are physically adjacent if the Veitch diagram is thought of as closed on itself horizontal-

ly. In this diagram, the terms may be amalgamated in twos or fours, which corresponds to the disjunctions of two or one variable. A minimal disjunctive normal form is obtained if all the 1's in the Veitch diagram are covered with the least number of the shortest products.

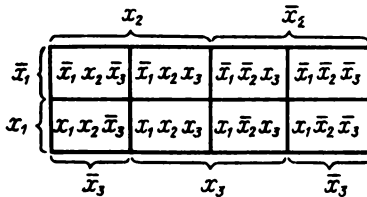


Fig. 5.2

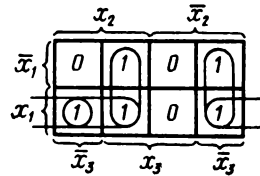


Fig. 5.3

**Example 2.** Find a minimal disjunctive normal form for the Boolean function

$$f(x_1, x_2, x_3) = \bar{x}_1 x_2 x_3 \vee x_1 x_2 \bar{x}_3 \vee x_1 x_2 x_3 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 \vee x_1 \bar{x}_2 \bar{x}_3$$

The Veitch diagram for this function appears in Fig. 5.3. With this diagram, two alternatives exist for combining the 1's into groups. Each alternative has a minimal disjunctive normal form of its own:

$$f(x_1, x_2, x_3) = x_1 \bar{x}_3 \vee x_2 x_3 \vee \bar{x}_2 \bar{x}_3$$

$$f(x_1, x_2, x_3) = x_1 x_2 \vee x_2 x_3 \vee \bar{x}_2 \bar{x}_3$$

The Veitch diagram for a Boolean function of four variables is shown in Fig. 5.4. In this diagram the squares in the lower-

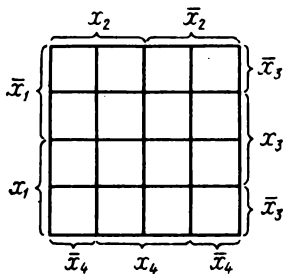


Fig. 5.4

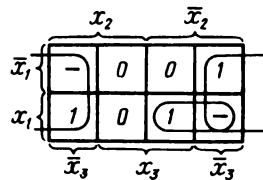


Fig. 5.5

most and topmost rows are physically adjacent if the diagram is thought of as closed on itself vertically. In this case, the basic conjunctions may be amalgamated in twos, fours and eights.

In conclusion of the chapter, we shall consider a procedure for simplifying incompletely defined Boolean functions which describe

logic circuits with so-called *forbidden combinations of input signals*. In this case, the function is arbitrarily defined on the sets of variables on which it has not been originally defined. In minimizing such functions, the forbidden combinations of inputs are assigned such values of the original function that will lead to its simplification.

**Example.** Minimize the Boolean function of three variables defined by a Veitch diagram (Fig. 5.5) for only six orderings of binary arguments. The undefined values are shown by dashes in the respective squares of the diagram (see Fig. 5.5).

Putting the function equal to unity for the orderings for which it has not been defined, the following minimal disjunctive normal form is obtained for the original function:

$$f(x_1, x_2, x_3) = \bar{x}_3 \vee \bar{x}_1 \bar{x}_2$$

## Chapter VI

# Basic Functional Elements of Electronic Digital Computers

### 6.1. Computer Elements

**6.1.1. General.** The basic functional elements of electronic digital computers are logical elements, storage elements (such as flip-flops and delay lines), and special-purpose elements. These elements are the minimal building blocks of an electronic digital computer.

*Logical elements* are computer elements which perform non-trivial logical functions (that is, some input or a combination of inputs produces something new as an output). As a rule, they have one output and as many inputs as there are input variables or arguments in the function being implemented. A set of logical elements (a system of elements) must be functionally complete. A set of logical elements is complete if all elements needed to implement the functions assigned to the system may be derived from them. If logical elements are based on transistors, flip-flop storage elements need not be included in the set of elements as separate entities, because they may be derived from the simpler logical elements.

A system of elements, apart from being functionally complete, must be physically complete; that is, it should include special elements needed to condition the waveform and amplitude of signals to standard limits. These special elements are amplifiers and shaping circuits. A system of elements satisfying the requirement of functional and physical completeness is called *technically complete*.

Depending on the requirements to be met, logical and storage elements may be based on a variety of physical principles and various circuit components.

According to the manner in which binary variables are represented, all computer elements may be classed into level elements, pulse elements, and pulse-level elements. In level elements, a binary variable of value 1 is represented as a high voltage level,  $+E_1$  (usually referred to as logical 1 level), and a binary variable of value 0 as a low voltage level,  $+E_0$  (usually referred to as logical 0 level). As an alternative, a binary variable of value 1 may be represented as a low level,  $-E_1$ , and a binary variable of value 0 as a high level,  $-E_0$ . Both  $+E_0$  and  $-E_0$  are practically zero. A distinction of level elements is resistive coupling between elements.

In contrast, there is no resistive coupling in pulse elements. In them, a binary variable of value 1 is represented as a pulse of a specified polarity and height, and a binary variable of value 0, as no pulse. Level-pulse (or pulse-level) elements use both representations for binary variables.

So that a correct choice can be made from among the various standard elements, it is important to know their behaviour and main characteristics. These include the following:

(1) Compatibility of input and output signals. Compatibility refers to the ability of an element to operate into other elements in complex circuits without the need for special elements to match input and output signals.

(2) Loading capability in terms of fan-in and fan-out. The fan-out defines the number of the inputs of other elements that can be connected to the output of a given element, provided that the distortion of the output signal will be within the tolerance limits. The fan-in gives the number of inputs that can be connected to a given logical circuit.

(3) Noise immunity, that is, the capability of an element to ignore noise at both input signal levels. Noise immunity can be expressed in terms of noise margin, which is the noise-to-signal ratio.

(4) The maximum clock rate or the minimum clock (or bit) time, defined as the minimum time interval between the instants when a logical element changes state, during which all transients completely die out.

(5) Capability for signal shaping, that is, for causing the signals that pass through to retain their standard waveform.

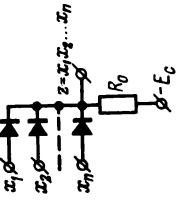
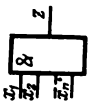
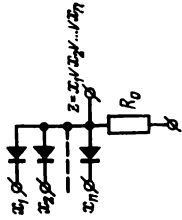

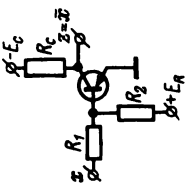

(6) Operating range determined by the limits of performance of the circuit components that make up a given computer element.

Other characteristics of importance are reliability, interchangeability, standardization, output signal delay time, and some others.

**6.1.2. Semiconductor logical elements.** The basic components in these elements are crystal diodes and transistors which may be discrete or formed in a monolithic integrated circuit. The schematics, operators, truth tables, the functions implemented, and diagram symbols of logical elements built around crystal diodes and *P-N-P* transistors are summarized in Table 6.1.

*Crystal diodes* offer a convenient means for implementing two Boolean functions, disjunction (OR) and conjunction (AND), because the resistance of crystal diodes changes suddenly upon polarity reversal of the applied voltage. Although diode logical circuits are passive, that is, they do not amplify the input signal, they offer a number of advantages, such as small size, low power drain and short switching time.

Table 6.1

Nos.	Schematic	Symbol in signal-flow diagrams	Truth table	Operator																																			
1			<table><thead><tr><th><math>x_1</math></th><th><math>x_2</math></th><th>...</th><th><math>x_n</math></th><th><math>z</math></th></tr></thead><tbody><tr><td>0</td><td>0</td><td>...</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>...</td><td>1</td><td>0</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr><tr><td>—</td><td>—</td><td>—</td><td>1</td><td>—</td></tr><tr><td>0</td><td>1</td><td>...</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>...</td><td>1</td><td>1</td></tr></tbody></table>	$x_1$	$x_2$	...	$x_n$	$z$	0	0	...	0	0	0	0	...	1	0	—	—	—	—	—	—	—	—	1	—	0	1	...	1	0	1	1	...	1	1	$D_c(x_1, x_2, \dots, x_n) =$ $= x_1 x_2 \dots x_n$
$x_1$	$x_2$	...	$x_n$	$z$																																			
0	0	...	0	0																																			
0	0	...	1	0																																			
—	—	—	—	—																																			
—	—	—	1	—																																			
0	1	...	1	0																																			
1	1	...	1	1																																			
2			<table><thead><tr><th><math>x_1</math></th><th><math>x_2</math></th><th>...</th><th><math>x_n</math></th><th><math>z</math></th></tr></thead><tbody><tr><td>0</td><td>0</td><td>...</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>...</td><td>1</td><td>1</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr><tr><td>—</td><td>—</td><td>—</td><td>1</td><td>1</td></tr><tr><td>0</td><td>1</td><td>...</td><td>1</td><td>1</td></tr></tbody></table>	$x_1$	$x_2$	...	$x_n$	$z$	0	0	...	0	0	0	0	...	1	1	—	—	—	—	—	—	—	—	1	1	0	1	...	1	1	$D_d(x_1, x_2, \dots, x_n) =$ $= x_1 \vee x_2 \vee \dots \vee x_n$					
$x_1$	$x_2$	...	$x_n$	$z$																																			
0	0	...	0	0																																			
0	0	...	1	1																																			
—	—	—	—	—																																			
—	—	—	1	1																																			
0	1	...	1	1																																			
3			<table><thead><tr><th><math>x</math></th><th><math>z</math></th></tr></thead><tbody><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></tbody></table>	$x$	$z$	0	1	1	0	$T_1(x) = \bar{x}$																													
$x$	$z$																																						
0	1																																						
1	0																																						



4		<div> <div><math>x</math></div> <div><math>z</math></div> </div>	<div> <div><math>x</math></div> <div><math>z</math></div> </div>	$E_1(x) = x$
5		<div> <div><math>x_1</math></div> <div><math>x_2</math></div> <div><math>z</math></div> </div>	<div> <div><math>x_1</math></div> <div><math>x_2</math></div> <div><math>z</math></div> </div>	$T_d^c(x_1, x_2) = \overline{x_1 \vee x_2} = \bar{x}_1 \bar{x}_2$
6		<div> <div><math>x_1</math></div> <div><math>x_2</math></div> <div><math>z</math></div> </div>	<div> <div><math>x_1</math></div> <div><math>x_2</math></div> <div><math>z</math></div> </div>	$E_d^d(x_1, x_2) = x_1 \vee x_2$



Consider the design of basic diode logical circuits.

(1) The diode AND gate mechanizes the conjunction of  $n$  variables (No. 1 in Table 6.1). The output signal of the AND gate is 1 ( $E_1$ ) if and only if all inputs are 1 at the same time. With any other combinations of inputs, the output of the AND gate will be 0. This is because when all inputs are 1 ( $E_1$ ), all the diodes will be driven to cut-off (with only small leakage currents flowing through them), and the output will be 1 ( $E_1$ ), too. If at least one input is 0, ( $E_0$ ), the respective diode will be conducting and, since the forward resistance of the diode is negligible, the output of the AND gate will be 0 ( $E_0$ ).

(2) The diode OR gate mechanizes the disjunction of  $n$  variables (No. 2 in Table 6.1). It produces a 1 output ( $E_1$ ) whenever any one (or more or all) of its inputs are 1 ( $E_1$ ).

(3) The diode-transformer AND gate and the diode-transformer OR gate are pulse-level logical circuits. Each gate uses diodes and a pulse transformer wound on a ferromagnetic core with a nonrectangular hysteresis loop. These elements provide a good match between the output impedance of a circuit and the input impedance of the next stage and generate pulse signals of any polarity, if there is no d.c. coupling. Most often, electronic digital computers use pulse-level AND gates with one pulse transformer (Fig. 6.1a). The output of this gate will be 1 when the level input, 1, is pulled up to the logical 1 level, and the pulse input, 2, is fed a positive-going pulse. The diode-transformer OR gate is shown in Fig. 6.1b. The polarity of the output signal can be reversed by an appropriate connection of the transformer windings.

*Transistors* may serve as a basis for logical circuits performing any elementary Boolean functions. In these circuits the transistors operate as switches, that is, devices with two stable states, the ON state (the transistor is in the saturated region) and the OFF state (the transistor is in the cut-off region).

Consider the design of basic transistor logical circuits.

(1) The transistor NOT gate (or inverter) is the most commonly used logical element. It realizes the logical NOT (negation) function (see No. 3, Table 6.1). The circuit parameters of the NOT gate are chosen such that the transistor resides in any one of two stable states and the output signal carries a maximum power. When a logical 0 is applied to the inverter input, a positive voltage is impressed across the base-emitter junction of the transistor (from a voltage divider,  $R_1/R_2$ ), so that the transistor is driven to cut-off. With load connected, the collector voltage will then be a logical 1. If a logical 1 is applied to the inverter input, a negative voltage will be impressed across the base-emitter junction, the transistor will turn on, and a logical 0 will appear

at the output, because the resistance of a conducting transistor is zero very nearly. To improve noise immunity, it is the usual practice to clamp the low output level with a diode,  $D4$ , and a supply source,  $-E_1$  (Fig. 6.1c). The NOT gate can be speeded in turn-on and turn-off by applying feedback via diodes (diodes  $D1$ ,  $D2$  and  $D3$ ), to prevent the transistor from saturating. In some circuits, the necessary speed-up is obtained by placing a capacitor across  $R1$  in the input divider.

Apart from its use as a logical element, the transistor NOT gate may be employed as a power amplifier or a shaping circuit.

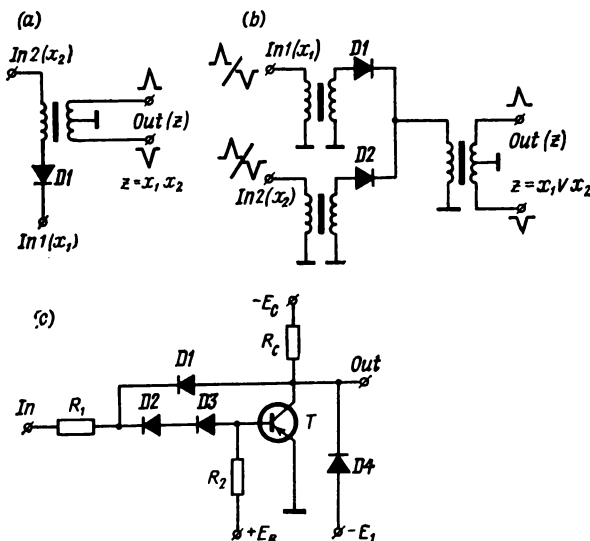


Fig. 6.1

(2) The emitter follower (when used alone) does not perform any logical function; it is used as a matching element or a power amplifier (see No. 4 in Table 6.1). A logical 1 at its input turns on the transistor, the voltage drop across the load resistor increases, and the output signal repeats (follows) the input signal.

(3) The NOT gate and the emitter follower can be modified by using common load resistors. Then, with parallel and series-connected transistors, four modifications of transistor logic can be derived with transistors of the same type of conduction. For simplicity, the number of inputs in the circuits shown at Nos. 5 through 8 in Table 6.1 has been reduced to two. The parallel gate performs the disjunction of the variables and negates the output if the transistors are connected in a NOT gate (No. 5 in Table 6.1),

or disjunction without negation if the transistors are connected in an emitter circuit (at No. 6 in Table 6.1). The series gate performs the conjunction of the inputs and the negation of the output if the transistors are connected in a NOT circuit (see No. 7 in Table 6.1) or conjunction without inversion if they are connected in an emitter circuit (see No. 8 in Table 6.1).

The symbols in the operators of the physical elements stand as follows.  $T$ 's and  $E$ 's indicate that the load resistor is placed in the collector or emitter circuit, respectively. The subscripts " $d$ " and " $c$ " indicate that the transistors are connected in parallel or in series and that the gate performs the function of disjunction or conjunction, respectively. The subscripts " $d$ " and " $c$ " of the

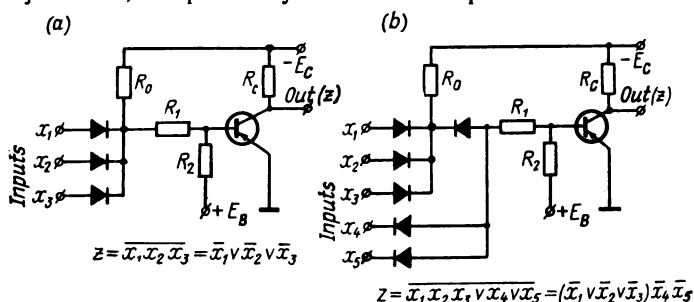


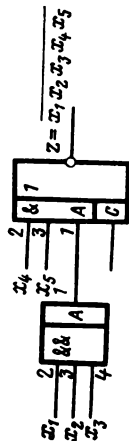
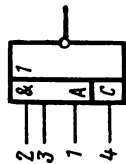
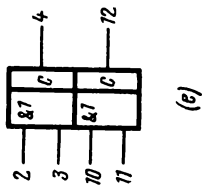
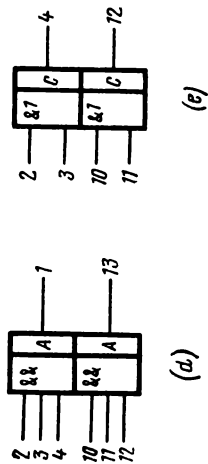
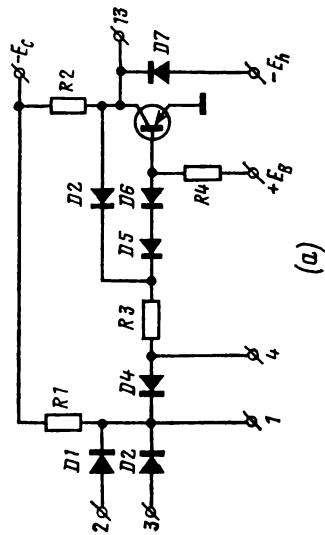
Fig. 6.2

symbol " $T$ " indicate that the output signal is inverted. The overscripts " $d$ " and " $c$ " on the symbols  $T$  and  $E$  indicate that the outputs bearing the same overscripts can be tied together. In such a case, " $d$ " designates disjunction and " $c$ ", conjunction. Thus, the operator notation gives an ample information from which the function being realized can be derived and the circuit implementing it may be constructed.

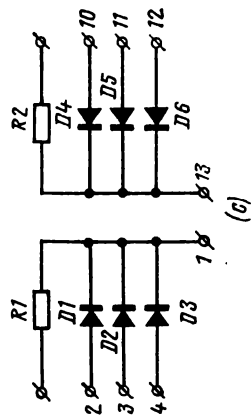
(4) Diode-transistor logical (*DTL*) elements are usually fabricated as general-purpose decision elements each consisting of a NOT gate and a one- or two-stage input diode logical circuit. Most often, the input circuit is a diode AND gate (Fig. 6.2a) or an AND/OR gate (Fig. 6.2b). *DTL* elements are more economical than transistor circuits. In the Soviet Union some computers, such as the Ural, use standardized Ural-10 series of modules built around *DTL* gates.

Physically, they are double-sided paper-base-laminate printed-circuit boards with discrete circuit components. Each module is driven by a pulse (change of voltage) going from a logical 0 ( $E_0 = -0.5$  to  $1.5$  V) to a logical 1 ( $E_1 = -6$  to  $-7.9$  V).

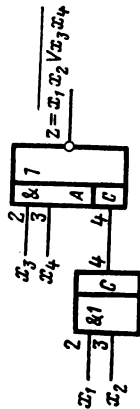
The key elements of the Ural-10 series are  $\mathcal{B}$  and  $\mathcal{D}$  modules which can mechanize any function, however complex.



(b)



(f)



(g)

A  $\bar{B}$  module whose circuit is shown in Fig. 6.3a and the logic symbol in Fig. 6.3b is a general-purpose logical element consisting of a non-saturating level NOT gate built around a drift transistor, and a two-stage input diode logic. It performs the nAND function when used alone, and the nAND/NOR function when used in conjunction with a  $\bar{A}$  module. Speed-up at turn-on and turn-off is provided by diode nonlinear feedback while the high ( $E_1$ ) level is clamped to enhance speed of operation and noise immunity. The variables are applied to inputs 2 and 3, and the output signal is taken from terminal 13. Inputs 1 and 4 are used solely for connecting a  $\bar{A}$  module.

A  $\bar{A}$  module whose circuit is shown in Fig. 6.3c and the logic symbol in Fig. 6.3d and e, is a diode circuit intended to augment the capabilities of  $\bar{B}$  modules. The  $\bar{A}$  module can be connected to a  $\bar{B}$  module in any one of two ways. If the objective is to increase the number of AND inputs, the  $\bar{A}$  module is connected as shown in Fig. 6.3f. In this case, the resistors  $R_1$  and  $R_2$  of the  $\bar{A}$  module are not connected to the  $-E_c$  source. If it is sought to increase the number of OR inputs, the connection is as shown in Fig. 6.3g. Now the resistors  $R_1$  and  $R_2$  must be connected to the  $-E_c$  source. Other types of connection will upset conditions for proper matching between the  $\bar{B}$  and  $\bar{A}$  modules.

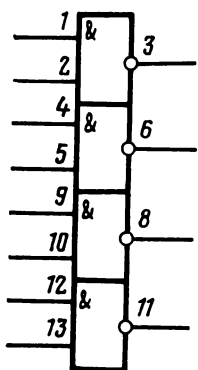
**6.1.3. Integrated logical circuits.** Integrated circuits have markedly improved reliability and reduced the size and power requirements of computers. At present, a wide range of integrated circuits are available commercially, including sets of decision, storage and support elements intended for use in the various assemblies and units of electronic digital computers. To illustrate, we shall examine K155 series elements.

The K155 series comprises ten modifications of integrated-circuit transistor-transistor logic (*TTL*) modules differing in the logical functions performed, fan-in/fan-out, and the count of elements per can. Their logic symbols are shown in Fig. 6.4a, b, c, d, e, f and g. The circuits of some (basic) modules appear in Fig. 6.4h, i and j. Those of the remaining modules only differ in the number of inputs.

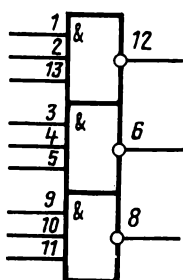
The logical capabilities of the modules shown in Fig. 6.4e and f can be augmented by the use of the module shown in Fig. 6.4g, when it is connected as shown in Fig. 6.4k.

The K155 series modules operate on  $E_c = +5$  V. The drive is a pulse (voltage change) going from low ( $E_{0\max} = 0.4$  V) to high ( $E_{1\max} = 2.4$  V).

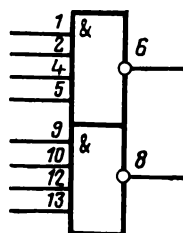
**6.1.4. Magnetic-core logical elements.** Of all the possible configurations of magnetic-core logic, we shall dwell on two types of bistable elements, namely, magnetic core-diode and magnetic core-transistor circuits. These elements have found wide use in the



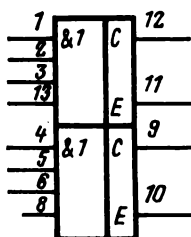
(a)



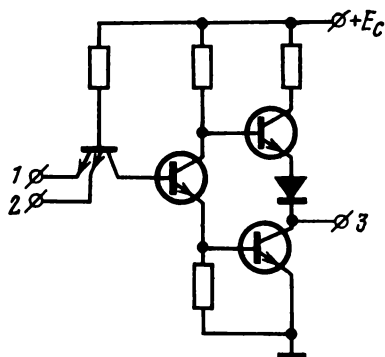
(b)



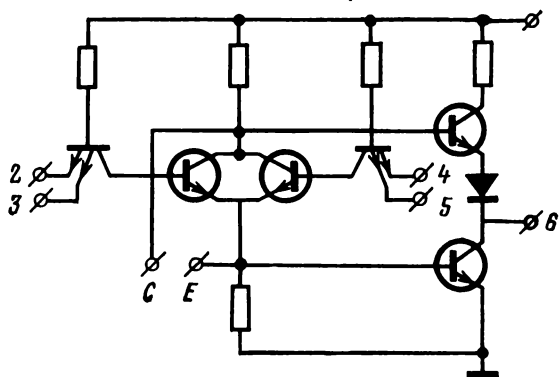
(c)



(g)



(h)



(j)

Fig. 6.4





slow-speed circuits of automatic control, telemetry and computers. The magnetic core of such an element, with windings applied to it, operates as a simple binary storage element. With no current pulses in the windings, the core will reside in one of two stable states (Fig. 6.5a), the positive residual flux-density state  $+B_r$ , designated binary 1, or the negative residual flux-density state  $-B_r$ , designated binary 0. Magnetic cores have been developed

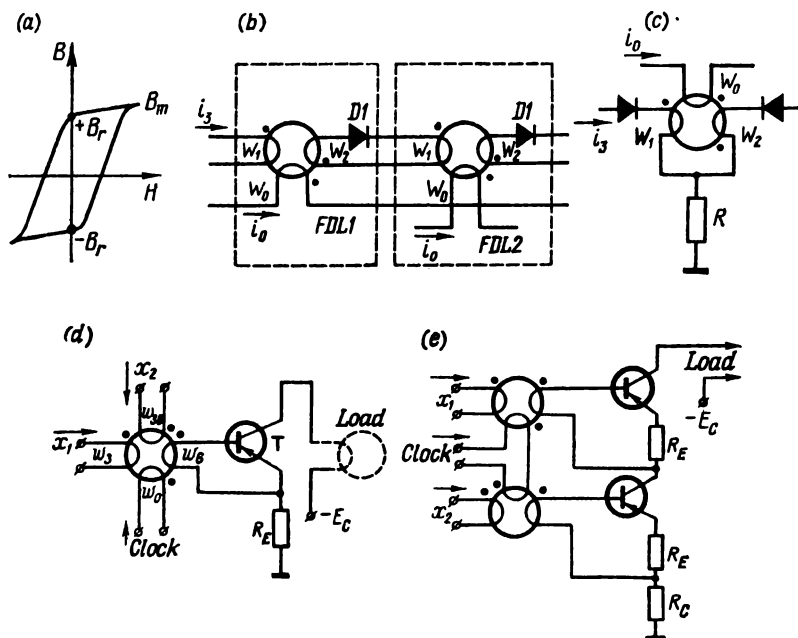


Fig. 6.5

in two basic forms, the metal strip types and the moulded ferrite.

**Ferrite core-diode logic.** The simplest ferrite core-diode element capable of information storage and transfer consists of a ferrite-core transformer showing bistable (or binary) properties owing to a rectangular hysteresis loop, and a diode placed in the output winding circuit. Connection of a ferrite core-diode element in a shift (or shifting) register is shown in Fig. 6.5b.

An operating cycle of a ferrite core-diode element consists of two periods, the write period and the read period. During the write period, if the input is a 1, the write signal applied to the input winding  $w_1$  sets the core to the 1 ( $+B_r$ ) state. The emf induced in the winding turns off the diode, and no current is allowed to flow to the load. During the read period, a clock pulse

applied to the winding  $w_0$  from a clock generator resets the core to the 0 ( $-B_r$ ) state. As a result, the emf induced in the winding  $w_2$  turns on the diode, and the information signal is allowed to reach the input winding of the load element. The emf induced in the output winding may be defined as

$$V_s \approx -w_2 S (\Delta B_1 / \tau) \text{ volts}$$

where  $S$  = cross-sectional area of the core,  $\text{cm}^2$

$\tau$  = average time needed to change state of magnetization,  $\mu\text{s}$

$w_2$  = number of turns in the output winding

$\Delta B_1 = B_m + B_r$  = change in flux density in reading out a 1, teslas

The signals applied to the inputs of a ferrite core-diode element (the windings  $w_1$  and  $w_0$ ) must be shifted from each other for a time exceeding the maximum time required for the core to change state of magnetization.

If, during the write period, the information signal is a 0, no write signal will be applied to the input winding  $w_1$ , and the core will remain in the previous state. As a result, the read pulse will produce magnetization in the same sense with relatively little or no flux change, that is, with little or no change in the state of magnetization. If, however, the core has a hysteresis loop of poor squareness, a read-out of 0 may induce a noise emf in the output winding such that a false 1 may be written into the load core. The noise emf may be defined as

$$V_n \approx -w_2 S \frac{B_m - B_r}{\tau} \text{ volts}$$

If the change in flux density is expressed in terms of the squareness ratio,  $k_{sq} = B_r/B_m$ , of the core, we shall get

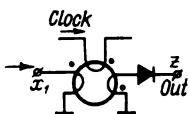
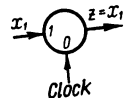
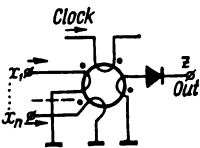
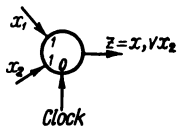
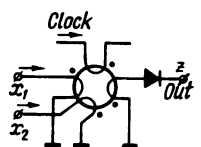
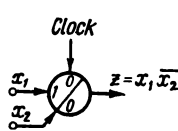
$$V_n \approx -w_2 S B_m (1 - k_{sq})$$

As is seen, the noise level decreases with increasing squareness ratio,  $k_{sq}$ .

A ferrite core-diode element used as a logic has several write windings. Two write windings connected in opposition implement the inhibit function, and the same two write windings connected aiding will mechanize the logical OR function. In the latter case, an ordinary diode OR function may be connected to the input winding.

The logic symbols and operators of the ferrite core-diode elements realizing the delay, OR and inhibit functions (in the order given) are summarized in Table 6.2.

Table 6.2

Nos.	Simplified circuit	Logic symbol	Truth table	Operator															
1			<table border="1"> <tr> <th><math>x_1</math></th> <th><math>z</math></th> </tr> <tr> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> </tr> </table>	$x_1$	$z$	0	0	1	1	$F_1(x_1) = x_1$									
$x_1$	$z$																		
0	0																		
1	1																		
2			<table border="1"> <tr> <th><math>x_1</math></th> <th><math>x_2</math></th> <th><math>z</math></th> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </table>	$x_1$	$x_2$	$z$	0	0	0	0	1	1	1	0	1	1	1	1	$F_{OR}(x_1, x_2) = x_1 \vee x_2$
$x_1$	$x_2$	$z$																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	1																	
3			<table border="1"> <tr> <th><math>x_1</math></th> <th><math>x_2</math></th> <th><math>z</math></th> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </table>	$x_1$	$x_2$	$z$	0	0	0	0	1	0	1	0	1	1	1	0	$F_{INHBT}(x_1, x_2) = x_1 \bar{x}_2$
$x_1$	$x_2$	$z$																	
0	0	0																	
0	1	0																	
1	0	1																	
1	1	0																	

In a simple ferrite core-diode element information might flow in the reverse direction. That is, on reading out a 1 written into, say, the core  $M_2$ , an emf might be induced in the winding, capable of switching the core  $M_1$ . To eliminate the backward flow of information and to enhance noise immunity, resort is made to special measures. For example the buffering action to eliminate the backward flow of information can be provided by a common resistor,  $R$  (Fig. 6.5c), placed in the input and output circuits. During read-out, no current will then flow in the input winding  $w_1$  because the voltage drop across  $R$  due to the current in the output circuit will be balanced by the emf induced across the winding  $w_1$ . The same purpose may be served by arranging the winding  $w_1$  to have a fewer number of turns than the winding  $w_2$  has.

For noise immunity, the signal applied to the inhibit input must be of a longer duration than that existing at the ordinary input, and the inhibit winding must carry a greater number of turns than the usual write winding does.

**Ferrite core-transistor elements.** As with a ferrite core-diode element, a simple ferrite core-transistor element can store and transfer binary information.

It consists of a ferrite-core transformer and a transistor placed in the output winding circuit (Fig. 6.5d). Ferrite core-transistor elements come in a variety of circuit configurations whose complexity (provision of bias, feedback and the like) is mainly determined by the service requirements (speed of operation, working temperature range and fan-out).

A ferrite-core transistor element operates in two phases. During the write period, the transistor blocks the passage of noise to the load. During the read period, when a 1 is read out, it cooperates with the core to produce an output signal which is transferred into the input windings of similar elements used as load, but blocks the reverse flow of information. In comparison with the diode variety, ferrite core-transistor elements need clock pulses of a lower power and are distinguished by a better noise immunity and speed of operation.

A simple ferrite core-transistor element can perform the inhibit function if it has two write windings connected in opposition. If an appropriate number of windings is connected aiding, the element will perform the logical OR function. The number of inputs for an OR gate using ferrite cores and transistors need not be limited, provided the write signals are applied over different circuits and at different clock times. However, the number of inputs should be limited for better noise immunity if there is a likelihood of several write signals coming at the same clock time. The logic symbols of the basic ferrite core-transistor elements performing the logical delay, inhibit and OR functions are the same as for ferrite core-diode elements (see Table 6.2).

The AND function can be implemented with two ferrite core-transistor elements by series-connecting their output circuits (Fig. 6.5e).

**6.1.5. Storage elements and their basic characteristics.** The storage (or memory) elements used in switching circuits with memory (sequential circuits) are various delay lines and flip-flops. Abstractedly, they are all Moore's automata and satisfy the following requirements: (1) they have two internal states, one symbolized as binary 1 and the other as binary 0; (2) for each internal state there is a specific output signal from which the internal state can readily be recognized. For convenience, the same letter symbols will be used to designate the internal states and output signals of a storage element.

To define the states of basic storage elements, the following notation will be adopted:

$q(t)$  is the input signal;

$Q(t)$  and  $Q(t+1)$  are the states of the storage element at times  $t$  and  $(t+1)$ , respectively;

$q_s$ ,  $q_r$ ,  $q_t$  are the set, reset and trigger (change or toggle) inputs, respectively.

The truth table for a basic switching circuit (a flip-flop) may be constructed, recalling that:

(1) a logical 1 applied to the reset input switches the circuit to the reset (0) state;

(2) a logical 1 applied to the set input switches the circuit to the set (1) state;

(3) simultaneous application of logical 1 signals to the set and reset inputs must be avoided because the output stage will be uncertain. In the truth table, the output squares corresponding to such forbidden input combinations are labelled with a dash;

(4) a logical 1 applied to the change (trigger or toggle) input causes the flip-flop to change state from set to reset or from reset to set;

(5) a logical 0 input signal will not change the state of the flip-flop.

The truth tables for basic sequential circuits (flip-flops) satisfying the conditions formulated above are given in Tables 6.3, 6.4, 6.5 and 6.6.

Table 6.3

$q(t)$	$Q(t)$	$Q(t+1)$
0	0	0
0	1	0
1	0	1
1	1	1

Table 6.4

$q_t(t)$	$Q(t)$	$Q(t+1)$
0	0	0
0	1	1
1	0	1
1	1	0

Table 6.5

$q_r(t)$	$q_s(t)$	$Q(t)$	$Q(t+1)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	—
1	1	1	—

Table 6.6

$q_i(t)$	$q_r(t)$	$q_s(t)$	$Q(t)$	$Q(t+1)$
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	—
0	1	1	1	—
1	0	0	0	1
1	0	0	1	0
1	0	1	0	—
1	0	1	1	—
1	1	0	0	—
1	1	0	1	—
1	1	1	0	—
1	1	1	1	—

The basic switching circuits under consideration have a complete transition system and a complete output system. A complete transition system means that for each of the four available pairs of states there is an input signal that will initiate the right transition. A complete output system means that in each state the circuit furnishes an output signal distinguishable from output signals in any other states.

For the purpose of synthesis, switching circuits may alternatively be specified with a transition matrix of the form:

	$x_1$	$x_2$	...	$x_l$	...	$x_n$
0-0	$b_{00}^1$	$b_{00}^2$		$b_{00}^l$		$b_{00}^n$
0-1	$b_{01}^1$	$b_{01}^2$		$b_{01}^l$		$b_{01}^n$
1-0	$b_{10}^1$	$b_{10}^2$		$b_{10}^l$		$b_{10}^n$
1-1	$b_{11}^1$	$b_{11}^2$		$b_{11}^l$		$b_{11}^n$

The transition matrix has four rows, which corresponds to a basic switching circuit with a complete transition system. The number of columns is the same as that of inputs. The matrix coefficient  $b_{kl}^i$  takes on the value of the signal existing at the  $l$ th input, which causes a transition from the  $k$ th to the  $l$ th state. The coef-

ficient  $b_{kl}^i$  will be indeterminate, if a transition has no dependence on input information. Using these rules, the following transition matrices can be formed from the respective truth tables.

(a)		(b)		(c)		
$q$		$q_t$		$q_r$	$q_s$	
0-0	0	0-0	0	0-0	$b_{00}^r$	0
0-1	1	0-1	1	0-1	0	1
1-0	0	1-0	1	1-0	1	0
1-1	1	1-1	0	1-1	0	$b_{11}^s$

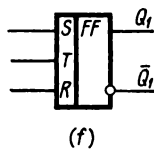
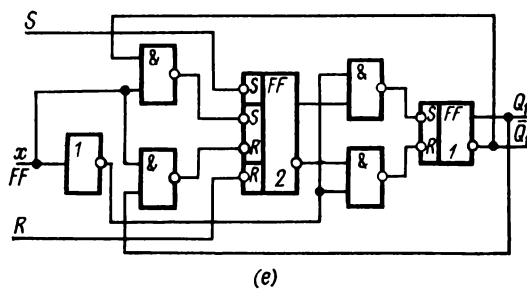
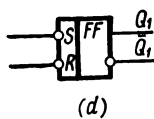
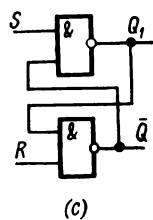
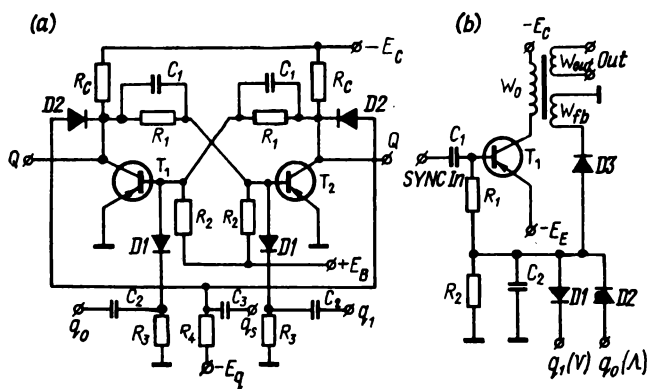
(d)			
	$q_t$	$q_r$	$q_s$
0-0	0	$b_{00}^r$	0
0-1	$b_{01}^t$	0	$\bar{b}_{01}^t$
1-0	$b_{10}^t$	$\bar{b}_{10}^t$	0
1-1	0	0	$b_{11}^s$

**6.1.6. Examples of typical flip-flop circuits.** According to the manner in which the output signals are represented, flip-flops may be classed into static and dynamic. As with level logical elements, the output signals of static flip-flops appear as changes in voltage level (from high to low or from low to high). Dynamic flip-flops, in contrast to pulse logical elements, present their output signals as a continuous sequence of pulses for a 1 (or true) state and as no pulse for a 0 (or false) state.

Figure 6.6a shows the well-known *basic flip-flop* which has all the three inputs. This is a direct-coupled Eccles-Jordan circuit made up of two NOT gates connected so that the input of one gate is coupled to the output of the other (that is, with their outputs cross-coupled). The RC-networks and diodes at the flip-flop input are triggering circuits. In the absence of trigger signals, the flip-flop will reside in one of two stable states, provided the inequality  $k < 1$  is satisfied, where  $k$  is the open-loop current or voltage gain.

Let the flip-flop be initially in the 0 state. Then transistor  $T_1$  will be conducting and transistor  $T_2$  will be at cut-off. If, now, a negative trigger signal is applied to the input  $q_1$ , it will turn on the transistor  $T_2$ , its collector will go high and pull up the base of transistor  $T_1$ , thereby cutting it off. The negative voltage





**Fig. 6.6**

(the logical — 1 signal) from the collector of the cut-off transistor will then be applied to the base of  $T_2$  to maintain it in the conducting state. All of these steps involved in a transition from state to state proceed regeneratively. The logical 1 appearing at the collector of  $T_1$  indicates that the flip-flop is in the 1 state.

Fan-out and reliability can be enhanced by using an inverter or an emitter follower at the output of the flip-flop. Speed-up can be provided in the same manner as in, say, an inverter (see Fig. 6.1c), that is, by using nonlinear diode feedback.

Figure 6.6b shows the circuit of a two-input *dynamic flip-flop* with a storage capacitor. This flip-flop operates as follows.

Assume that initially transistor  $T_1$  is cut off by a positive voltage,  $+E_E$ , applied to its base relative to the emitter. Timing (clock) pulses cannot get through to the output because they are lower in amplitude than  $E_E$ . This is the 0 state for the flip-flop. The flip-flop is set to the 1 state by a negative "1" signal applied to the input  $q_1$ . When this happens, the storage capacitor  $C_2$  charges to  $-E_E$ , thereby pulling the base of the transistor to ground. Then a negative clock pulse will turn on the transistor, and a logic 1 signal will appear at the output, boosted and shaped by the transformer. From that instant on, the flip-flop will be generating a sequence of pulses due to positive feedback which replenishes the charge across the capacitor  $C_2$  via diode  $D_3$ . Arrival of a positive "1" signal at the input  $q_0$  will cause the capacitor  $C_2$  to discharge, the voltage at the base of  $T_2$  will rise to turn off the transistor, and clock pulses will no longer be able to get through to the output.

Of the many logically different flip-flops, present-day digital computers mainly use set-reset ( $RS$ ) flip-flops, trigger or toggle ( $T$ ) flip-flops, and reset-set-trigger ( $RST$ ) flip-flops.

Set-reset flip-flops are combinations of two NAND gates cross-coupled to provide positive feedback. A set-reset flip-flop is caused to change state by applying a "0" signal to the set ( $S$ ) or reset ( $R$ ) input (Fig. 6.6c and d).

Set-reset-trigger flip-flops are combinations of  $RS$  flip-flops and additional logical elements. The  $RST$  flip-flop shown in Fig. 6.6e and d operates as follows. When  $x = 0$ , information is copied from  $T_2$  to  $T_1$  in sign-and-magnitude form. When  $x = 1$ , information is copied from  $T_1$  to  $T_2$  in 1's complement form. This is why, after an  $RST$  flip-flop is set to 0 for  $x = 1$ , a logical 1 is written into  $T_2$ ; during the next bit (or clock) time (that is for  $x = 0$ ) the logical 1 will be copied back to  $T_1$ , and the  $RST$  flip-flop is thereby set to the "1" state ( $Q_1 = 1$ ). After a second signal ( $x = 1$ , then  $x = 0$ ) is applied to the  $x$ -input, the  $RST$  flip-flop will be switched to the "0" state ( $Q_1 = 0$ ).

## 6.2. Logical Design

**6.2.1. General.** Digital systems are built from standard building blocks with two states, called logical elements which include decision and storage elements. The art of organizing them is known as *logical design*, and the systems themselves implementing the functions of finite automata, including combinational and sequential varieties, are called *logical networks*.

The ultimate goal of logical design is to synthesize an optimal structure for the desired logical network. Conversely, the objective of logical-network analysis is to derive either Boolean functions in the case of a combinational network (a single-state switching circuit) or input-output tables in the case of sequential networks.

In many cases, the first step in logical design for automatic control systems and computers is a verbal description of how the desired logical network is to operate. The next step is to translate the verbal description into a suitable mathematical formalism such as Boolean (switching) functions or input-output (truth) tables. A third step is the selection of a set of logical elements for the desired logical network. As has been noted, an important requirement in this respect is that the logical elements should form a functionally and physically complete set. A further requirement is a minimum count of components necessary to realize the network. As regards combinational networks, the minimal-count requirement necessitates the presentation of the original switching function in its minimal disjunctive normal form.

Obtained through appropriate minimization, the minimal disjunctive normal form of the switching function describing the combinational network is transformed into a superposition of the operators of the logical elements chosen for the desired logical network. The number of basic operator symbols in the modelled switching functions will determine the total count of physical elements needed.

The next step in logical design is to draw up a block diagram for the logical network. The block diagram affords a picture of the switching function presented as the superposition of logical operators, and also shows the interconnections of the physical devices by equivalent interconnections of logical symbols representing these devices.

Before proceeding to the final stage of logical design — that of physical implementation, problems related to signal amplification and timing are tackled. Above all, a proper timing must be provided to ensure the correct transfer of information between combinational and memory networks (that is, between decision and storage elements).

With combinational networks (networks built from decision elements, that is, logical elements which have no property to store information), new information may be read in only after the previous signals have ceased and all the elements have been reset. Also, if there is a limit on the signal duration, it is required that signals to the inputs of a given element should come from the previous stages all at the same time. If the preceding stages delay signals for different times, suitable delay lines must be provided to equalize the time delays.

With sequential networks (logical networks incorporating storage elements), it is important that a read signal and a control signal be applied to a storage element (a flip-flop) at different

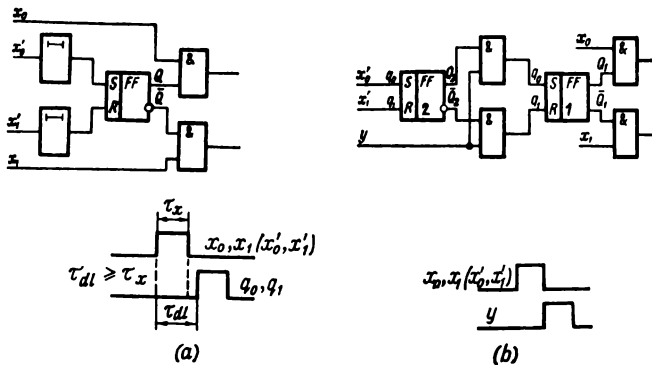


Fig. 6.7

instants. If a control signal is shaped according to the internal state of the flip-flop or if a read signal occurs at the same time with a control signal, the latter must be delayed for the duration of the read signal. The control signal may be delayed either by a suitable delay line when information is transferred within a single clock time (Fig. 6.7a), or by means of an additional flip-flop, when information transfer occupies two clock times (Fig. 6.7b). In the latter case, new information is written into temporary-storage flip-flop 2 and the previous information is read out of permanent-storage flip-flop 1 within the same clock time. During the second clock time, the control signal,  $y$ , causes the information to be copied from temporary-storage flip-flop 2 to permanent-storage flip-flop 1.

Before we take up some of the procedures involved in the synthesis of combinational networks, it is worth while considering their analysis.

Analysis is done in two stages. To begin with, a system of switching (Boolean) functions is derived from the block diagram

to bring out the relationships between the logical elements in the network. Each of these switching functions is implemented by one of the logical elements, and its input variables have the same symbols as the output variables of the preceding stage.

Then the operation of internal superposition (substitution) is consecutively applied to the system of switching functions until a function expressed solely in terms of input variables is derived.

**Example.** Derive the switching function mechanized by the logical network shown in Fig. 6.8.

The system of switching functions will have the form

$$z = y_1 \vee y_2, \quad y_1 = x_1 y_3, \quad y_2 = x_3 y_4, \quad y_3 = \bar{x}_2, \quad y_4 = \bar{x}_4$$

Hence,

$$z = x_1 \bar{x}_2 \vee x_3 \bar{x}_4$$

**6.2.2. Synthesis of combinational networks.** We shall follow the techniques used in the derivation of block diagrams, using the realization of simple switching functions by various physical devices. It will be assumed that the original function has been reduced to its minimal disjunctive normal forms by one of the methods described earlier.

So that the original switching function can be transformed into a superposition of logical operators, the minimal disjunctive normal form in some cases has to be recast so as to minimize the count of equipment needed.

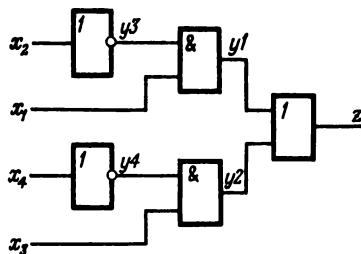


Fig. 6.8

### Single-output Networks

We shall first discuss networks using semiconductor devices. In the examples that follow we assume that the variables are present at the inputs together with their complements.

**Example 1.** Implement the function

$$z = \bar{x}_1 x_2 \vee x_1 \bar{x}_3 \vee \bar{x}_2 x_3$$

using various operators.

No preliminary transformations are needed in order to implement the function, using the operator  $E_c^d$ . Therefore, as a superposition of this operator, the function can be written as:

$$z = E_c^d(\bar{x}_1, x_2) \vee E_c^d(x_1, \bar{x}_3) \vee E_c^d(\bar{x}_2, x_3)$$

The circuit realizing it is shown in Fig. 6.9a.

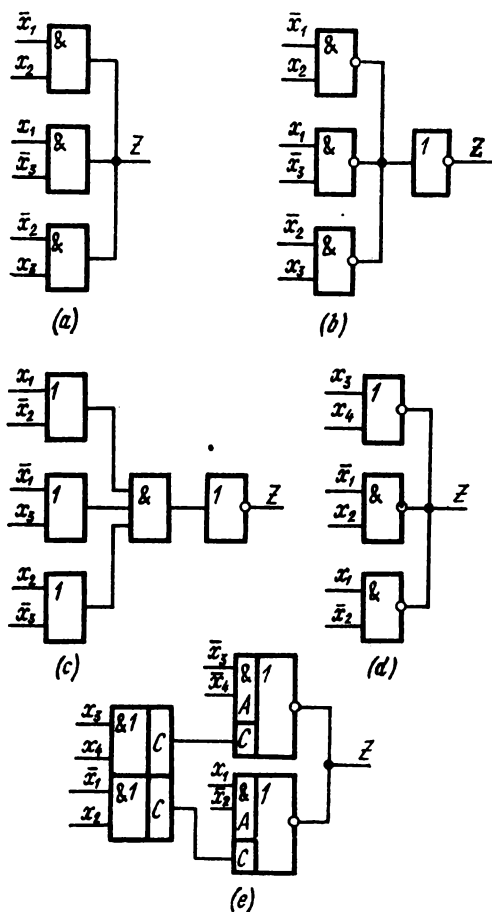


Fig. 6.9

In order to implement the given function with operators of the form  $T_c^c$  and  $T_1$ , it must be re-cast as

$$z = \overline{(x_1 x_2)(x_1 x_3)(x_2 x_3)}$$

Then, written as a superposition of operators, the function will be

$$z = T_1 [T_c^c(\bar{x}_1, x_2) T_c^c(x_1, \bar{x}_3) T_c^c(\bar{x}_2, x_3)]$$

The circuit implementing it is shown in Fig. 6.9b.

If the given function is to be mechanized with DTL elements whose operators are  $D_a$ ,  $D_d$  and  $T_1$ , the function must be re-ar-

ranged as

$$z = \overline{(x_1 \vee \bar{x}_2)(\bar{x}_1 \vee x_3)(x_2 \vee \bar{x}_3)}$$

whence in the form of a superposition of operators it can be written as

$$z = T_1 \{D_c [D_d(x_1, \bar{x}_2), D_d(\bar{x}_1, x_3) D_d(x_2, \bar{x}_3)]\}$$

The circuit mechanizing it has the form shown in Fig. 6.9c.

**Example 2.** Implement the function

$$z = x_1 x_2 \bar{x}_3 \bar{x}_4 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4$$

using transistor operators.

The realization of this function may be used as an example to show how the given function can effectively be simplified by factoring out the common multiplier and by reducing the expression inside the brackets to its expanded disjunctive normal form:

$$z = \bar{x}_3 \bar{x}_4 (x_1 x_2 \vee \bar{x}_1 \bar{x}_2) = \bar{x}_3 \bar{x}_4 (x_1 \vee \bar{x}_2)(\bar{x}_1 \vee x_2)$$

Using the operators  $T_c^c$  and  $T_d^c$  for implementation, we obtain

$$z = T_d^c(x_3, x_4) T_c^c(\bar{x}_1, x_2) T_c^c(x_1, \bar{x}_2)$$

The circuit implementing it is shown in Fig. 6.9d.

**Example 3.** Implement the function

$$z = x_1 x_2 x_3 \bar{x}_4 \vee \bar{x}_1 \bar{x}_2 x_3 \bar{x}_4 \vee x_1 x_2 \bar{x}_3 x_4 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 x_4$$

using Type B and Type D modules of the Ural-10 series.

The given function can be reduced to the form

$$\begin{aligned} z &= x_3 \bar{x}_4 (x_1 x_2 \vee \bar{x}_1 \bar{x}_2) \vee \bar{x}_3 x_4 (x_1 x_2 \vee \bar{x}_1 \bar{x}_2) = (x_1 x_2 \vee \bar{x}_1 \bar{x}_2) (x_3 \bar{x}_4 \vee \bar{x}_3 x_4) \\ &= \overline{(x_1 \bar{x}_2 \vee \bar{x}_1 x_2)} \overline{(x_3 x_4 \vee \bar{x}_3 \bar{x}_4)} \end{aligned}$$

This expression is a conjunction of two expressions identical in presentation, each of which can be mechanized with a B module and a half of a D module, with the conjunction implemented by combining the collector loads of the B modules. The resultant circuit configuration is shown in Fig. 6.9e.

Now we shall consider networks implemented with *bistable magnetic-core elements*. The following set of basic logical elements will be used to implement switching functions: a two-input inhibit gate (operator  $F_{INHBT}$ ), an OR gate (operator  $F_{OR}$ ), a one-bit delay element (operator  $F_1$ ) and a units generator (operator "1"). The original function will be represented as a superposition of the chosen operators, with every effort being taken to simplify it so as to minimize the count of equipment. The number of elements needed to implement the original function is the same as the number of operators in the presentation of the function.

To ensure the correct transfer of information, signals must be applied to the inputs of an element all at the same clock time. Should any signal be generated one clock time before another, it must be delayed for precisely one clock time by a delay element. In operator notation, this circumstance will be shown by placing the variable to be delayed under the sign of the operator  $F_1$ . Hence, the following general rule may be stated for a check on the operator equation for the original function: the variables standing under the sign of an operator must be generated all at the same clock time.

The inversion of a variable can be implemented according to the relation

$$z = \bar{x}_1 = 1 \cdot \bar{x} = F_{INHBT}(1, x) \quad (6.1)$$

Conjunction of variables can be realized according to the tautology

$$x_1^{\sigma_1} x_2^{\sigma_2} \dots x_n^{\sigma_n} = 1 \left( \overline{\bar{x}_1^{\sigma_1} \vee \bar{x}_2^{\sigma_2} \vee \dots \vee \bar{x}_n^{\sigma_n}} \right) \quad (6.2)$$

If a conjunction involves only one variable without its inversion, for example,  $x_k$ , it will be more economical to implement it according to the tautology

$$\bar{x}_1 \bar{x}_2 \dots \bar{x}_k \dots \bar{x}_n \\ = x_k (x_1 \vee x_2 \vee \dots \vee x_{k-1} \vee x_{k+1} \vee \dots \vee x_n) \quad (6.3)$$

The realization of a function containing a common multiplier can materially be simplified by factoring it out, especially if the expression in the parentheses is an implication. For example, by simplifying the function

$$z = x_1 x_2 \vee x_2 \bar{x}_3$$

we get

$$\begin{aligned} z &= x_2 (x_1 \vee \bar{x}_3) = x_2 (\overline{\bar{x}_1 x_3}) = x_2 (\overline{x_3 \bar{x}_1}) \\ &= F_{INHBT}[x_2(x_3, \bar{x}_1)] = F_{INHBT}[F_1(x_2), F_{INHBT}(x_3, x_1)] \end{aligned}$$

The circuit mechanizing this function is shown in Fig. 6.10a. For simplicity, the clock pulses are not shown in the diagrams.

**Example.** Implement the function

$$z = x_1 x_2 \bar{x}_3 \vee \bar{x}_1 x_2 x_4$$

with bistable magnetic-core elements.

To begin with, the given function can be simplified to the form

$$z = x_2 \overline{(x_1 \bar{x}_3 \vee x_4 \bar{x}_1)}$$



Then, as a superposition of logical operators, the original function may be represented as follows:

$$\begin{aligned} z &= F_{INHBT} [x_2 (\bar{x}_1 \bar{x}_3 \vee x_4 \bar{x}_1)] = F_{INHBT} [F_1(x_2), \\ &F_{INHBT}(1, x_1 \bar{x}_3 \vee x_4 \bar{x}_1)] = F_{INHBT} \{F_1[F_1(x_2)], \\ &F_{INHBT}[1, F_{OR}(x_1 \bar{x}_3, x_4 \bar{x}_1)]\} = F_{INHBT} \{F_1[F_1[F_1(x_2)]], \\ &F_{INHBT}[1, F_{OR}[F_{INHBT}(x_1, x_3), F_{INHBT}(x_4, x_1)]]\} \end{aligned}$$

The circuit implementing this function appears in Fig. 6.10b.

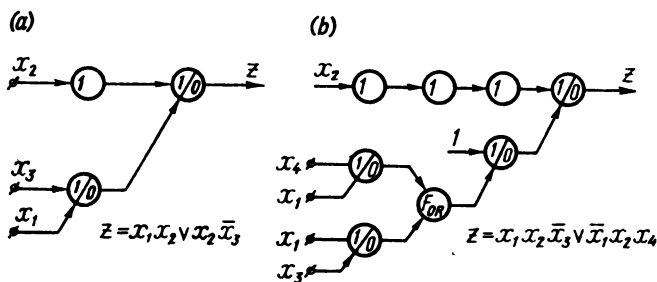


Fig. 6.10

### Multi-output Networks

A multi-output combinational network implements as many switching functions as there are outputs on the network. It can be constructed in any one of several ways.

(1) Each constituent switching function can be implemented independently. However, this approach is uneconomical in many cases.

(2) The number of elements needed can materially be reduced by expressing one function in terms of another if the two have common terms. For example, for three functions of the form

$$\begin{aligned} z_1 &= \bar{x}_1 \bar{x}_2 \bar{x}_3 \vee \bar{x}_1 \bar{x}_2 x_3 \\ z_2 &= \bar{x}_1 \bar{x}_2 \bar{x}_3 \vee \bar{x}_1 \bar{x}_2 x_3 \vee x_1 x_2 x_3 \\ z_3 &= \bar{x}_1 \bar{x}_2 \bar{x}_3 \vee \bar{x}_1 \bar{x}_2 x_3 \vee x_1 x_2 x_3 \vee x_1 \bar{x}_2 x_3 \end{aligned}$$

we may write

$$\begin{aligned} z_2 &= z_1 \vee x_1 x_2 x_3 \\ z_3 &= z_2 \vee x_1 \bar{x}_2 x_3 \end{aligned}$$

(3) Sometimes, the given functions may be simplified by using one of them (after it has been derived) as an auxiliary input variable in deriving another function. The function to be thus used should be one permitting the simplest realization.

**Example.** Using the above technique, simplify the system of two functions

$$z_1 = \bar{x}_1 \bar{x}_2 \bar{x}_3 \vee \bar{x}_1 \bar{x}_2 x_3 \vee \bar{x}_1 x_2 \bar{x}_3$$

$$z_2 = \bar{x}_1 x_2 x_3 \vee x_1 \bar{x}_2 \bar{x}_3 \vee x_1 \bar{x}_2 x_3$$

to an expression calling for the least count of devices.

After minimization,

$$z_1 = \bar{x}_1 \bar{x}_2 \vee \bar{x}_1 \bar{x}_3$$

$$z_2 = x_1 \bar{x}_2 \vee \bar{x}_1 x_2 x_3$$

Using the function  $z_1$  as an input variable to derive  $z_2$ , we get

$$z_2 = \bar{x}_1 x_2 x_3 \bar{z}_1 \vee x_1 \bar{x}_2 x_3 \bar{z}_1 \vee x_1 \bar{x}_2 \bar{x}_3 \bar{z}_1$$

This function is not defined for eight sets of variables. Therefore, using the minimization technique for incompletely defined functions based on the Veitch diagram (Fig. 6.11), we obtain the following minimal form:

	$x_2$		$\bar{x}_2$		
					$\bar{x}_3$
$\bar{x}_1$	-	0	0	-	$\bar{x}_3$
	1	-	0	-	
$x_1$	0	-	-	1	$x_3$
	0	-	-	1	
	$\bar{z}_1$	$z_1$	$\bar{z}_1$		

Fig. 6.11

$$z_2 = \bar{z}_1 \bar{x}_2 \vee \bar{z}_1 \bar{x}_1 = \bar{z}_1 (\bar{x}_1 \vee \bar{x}_2)$$

which is simpler to realize than if it were implemented independently.

(4) A good deal of simplification can be obtained by use of the cascading method in which some of the elements in a network are employed to mechanize several functions at a time.

### 6.2.3. Logical design of sequential networks.

Most often, the logical design of sequential networks specified by state tables is carried out by the *canonical-form method*. With this method, the logical design of an arbitrary sequential network reduces to that of a combinational network. For the purpose of this method, the original sequential network is presented (Fig. 6.12) as consisting of two parts: a memory section made up of basic storage elements and a combinational section made up of basic logical (decision) elements.

In order to develop a block diagram of any sequential network, one must have a structurally complete set of elements, including basic storage elements with their state tables, and a functionally complete set of basic logical (decision) elements. If the basic storage elements are built from basic logical (decision) elements, they may be omitted from the structurally complete set.

The algorithm for the logical design of a sequential network by the canonical-form method will best be illustrated by an example.

**Example.** Develop a block diagram for a sequential network which, in response to a sequence of inputs  $x_1 = 1$ , returns a sequence of binary numbers identical with the binary states of storage elements 0, 2, 1, 3, 0, 2, etc.; and, in response to a sequence of inputs  $x_2 = 1$ , it returns a sequence 0, 1, 2, 3, 0, 1, etc. The inputs  $x_1 = x_2 = 0$  do not change the state of the network. Two inputs  $x_1 = 1$  and  $x_2 = 1$  are not allowed to occur simultaneously.

The sequential circuit has a total of four states. Therefore, for its realization it will be sufficient to have two storage elements with a complete set of transitions and outputs. These storage elements may be static  $T$  flip-flops. The state and output of the first  $T$  flip-flop will be designated  $Q_1$ , and those of the second,  $Q_2$ .

Noting the conditions under which the desired sequential network is to operate, we compile its state table (columns 1 through 6 in Table 6.7).

The excitation functions,  $q_{s1}$  and  $q_{s2}$ , of the flip-flops can be determined from their excitation matrices. Noting that  $q_{1s}$  and  $q_{s2}$

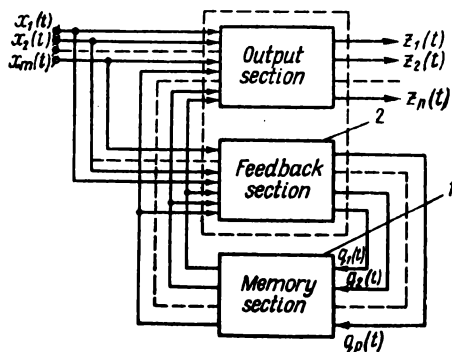


Fig. 6.12

Table 6.7

1	2	3	4	5	6	7	8
$x_1$	$x_2$	$Q_1(t)$	$Q_2(t)$	$Q_1(t+1)$	$Q_2(t+2)$	$q_{s1}$	$q_{s2}$
0	1	0	0	0	1	0	1
0	1	0	1	1	0	1	1
0	1	1	0	1	1	0	1
0	1	1	1	0	0	1	1
1	0	0	0	1	0	1	0
1	0	0	1	1	1	1	0
1	0	1	0	0	1	1	1
1	0	1	1	0	0	1	1
0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0
0	0	1	0	1	0	0	0
0	0	1	1	1	1	0	0

will be unity only if the respective flip-flop switches from the 0 to 1 state or the other way round, we fill columns 7 and 8 of Table 6.7.

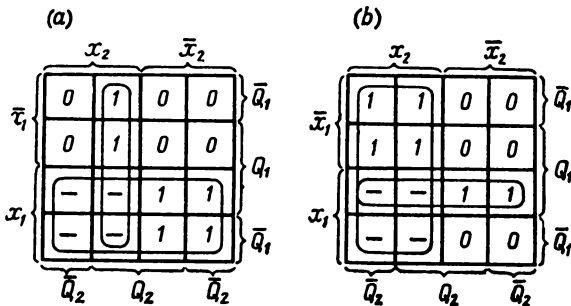


Fig. 6.13

Since for the sequential network in question the excitation functions of the flip-flops are partially specified (because the inputs  $x_1 = 1$  and  $x_2 = 1$  are never applied together), we may minimize the excitation functions with a Veitch diagram (Fig. 6.13a and b), whence it follows that

$$q_{s1} = x_1 \vee x_2 Q_2$$

$$q_{s2} = x_2 \vee x_1 Q_1$$

According to the statement of the problem, the output signals of the sequential network are identical with the internal states of the flip-flops,  $Q_1$  and  $Q_2$ .

Using the expressions for  $q_{s1}$  and  $q_{s2}$  and noting the rules for the correct transfer of information, we may depict the block diagram of the sought sequential network as shown in Fig. 6.14.

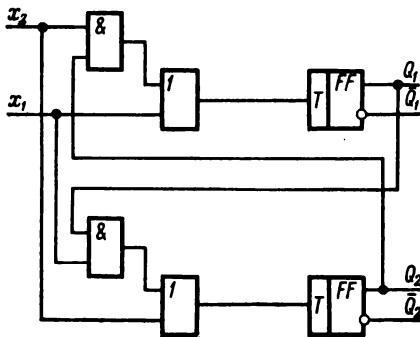


Fig. 6.14

### 6.3. Logic Assemblies of Electronic Digital Computers

**6.3.1. Registers.** The basic function of a register is to store a machine word, parts thereof, and some functional attributes. This function involves the operations of read-in, storage and read-out (or transfer). If a register additionally performs the operation of shift, it is called a *shift* (or shifting) register.

All operations in a register are performed in response to appropriate control or gate signals, which are as follows:

- $v_{ri}$  causes a word to be read into a register;
- $v_0$  clears the register, that is, sets it to zero;
- $v_{sr}$  and  $v_{sl}$  cause a word in the register to be shifted right or left, respectively;

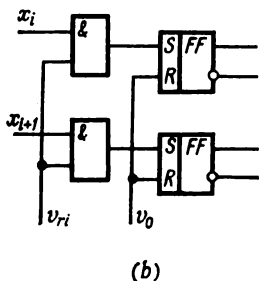
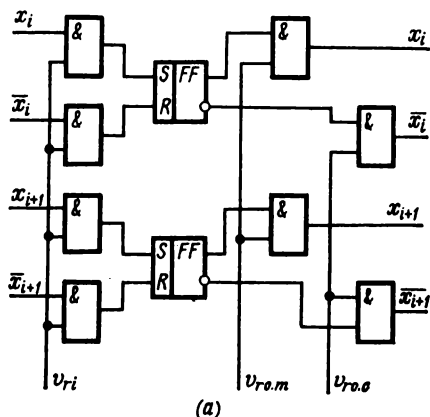


Fig. 6.15

$v_{ro.m}$  and  $v_{ro.c}$  cause a word to be read out of a register in sign-and-magnitude or 1's complement form, respectively.

Most commonly, registers are built from RS flip-flops (for a description of flip-flops as independent entities, refer to Sec. 6.1).

A new word may be read into a register either independently of the previous information stored if the register has inverting inputs (Fig. 6.15a), or only after the register has been cleared (Fig. 6.15b). Registers may be constructed to deliver a word to the output in parallel (a parallel register) as shown in Fig. 6.15 (in

sign-and-magnitude or 1's complement form), or serially (a serial register). In the latter case, the bits of a word are made available sequentially, one after another. In effect, this operation is a special case of shift, and a simple serial register is often called a shifting register.

As a rule, registers are built with a capability to shift a word one place at a time. If a word has to be shifted  $n$  places, the shift operation will have to be repeated  $n$  times. Where a word must be shifted any number of places over the same time interval, resort is made to a special assembly called a *shifter*.

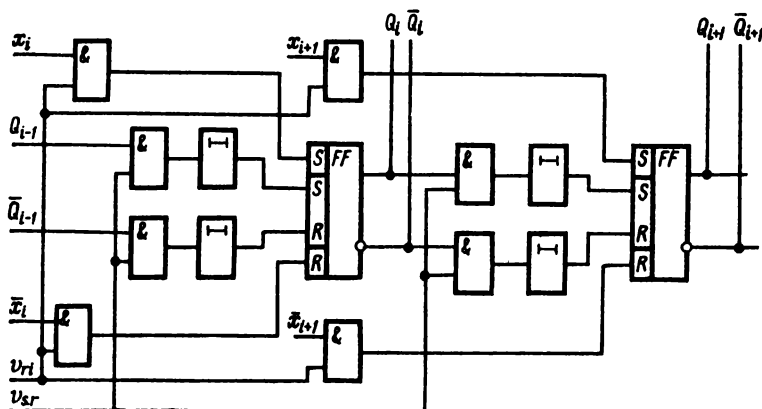


Fig. 6.16

*A shift register using pulse-level elements.* In block-diagram form, two stages of a shift register using pulse-level elements are shown in the diagram of Fig. 6.16. The pulse-level elements are AND gates performing read-in, read-out and shift. The shift capability is provided by connecting the outputs of each stage to the inputs of the next stage via an AND gate and delay lines. This organization results in a single register which uses transient storage during the shifting process. Figure 6.16 shows only the circuit that shifts a word to the right. According to the rules for the transfer of information, the delay time in the shift circuits is specified by the condition

$$\tau_{d,1} \geq \tau_{s,p}$$

where  $\tau_{d,1}$  is the delay provided by a delay line and  $\tau_{s,p}$  is the duration of shift pulses sufficient for the flip-flop to operate. The repetition period for shift pulses,  $T_s$ , is selected from the condition

$$T_s \geq \tau_{s,p} + \tau_{d,1}$$

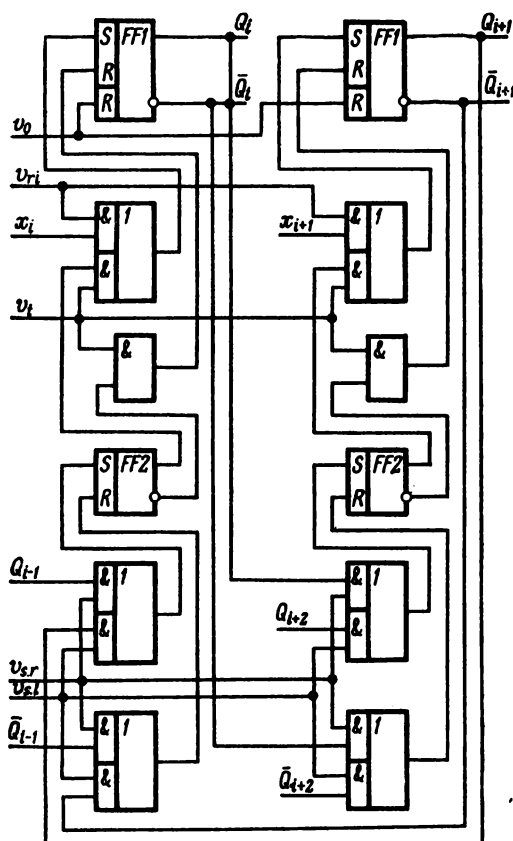


Fig. 6.17

*A shift register using level elements.* As is seen from the block diagram of Fig. 6.17, this is a "double" register, operating in basically two steps. Each stage consists of two flip-flops, the permanent-storage flip-flop  $FF1$ , and the temporary-storage flip-flop  $FF2$ . The outputs of  $FF1$  are connected via AND gates to the like inputs of  $FF2$  in the next stage, while the outputs of  $FF2$  are coupled via AND gates to the inputs of  $FF1$  in the same stage.

The shifting process takes place in basically two steps. In the first step, a  $v_{s,r}$  or a  $v_{s,l}$  pulse causes a word to be transferred with a shift from  $FF1$  into  $FF2$ . In the second step, a  $v_i$  pulse causes the word to be transferred from  $FF2$  into the corresponding bit positions of  $FF1$ , directly (without shifting left or right).

Among the advantages of the double-register scheme is high reliability because information is never stored in the transient state. Also, each gate signal may start immediately after the previous one has terminated.

**6.3.2. Counters.** This computer assembly has several stable states. It is mainly used to count pulses applied to its input. In an electronic digital computer, counters generate word (number) addresses in the storage unit and instruction addresses in the control unit, and special sequences of control or gate pulses, count the number of cycles for some operations, etc.

Counters may be classed in any one of several ways. For example, they are classed into scale-of-two, scale-of-ten, etc. counters according to the radix or base chosen for number representation or the number of states that a counter has. Or they may be classed as adding (up-counters), subtracting (down-counters) and reversible (bidirectional) according to the arithmetic operation they perform. A further classification is according to the manner in which the carry operation is organized (counters with a cascaded carry, ripple-through carry, block carry and partial-block carry).

The basis of any counter is a one-stage binary counter which produces an output pulse for every two input pulses (it is said to take a modulo 2 sum) and is built around a variety of flip-flops, mostly  $T$  flip-flops. The state of a binary counter at time  $(t + 1)$  is a function of its own state and its input at time  $t$ .

In the logical design of a counter, it is treated as a finite-state machine with a single input and  $2^n$  outputs, where  $n$  is the maximum number of bits that the counter can hold. The output signals of the counter must be in binary form and in a one-to-one correspondence with the states of the storage elements in the counter.

Table 6.8

1	2	3	4	5	6	7	8	9	10
$x$	$Q_1(t)$	$Q_2(t)$	$Q_3(t)$	$Q_1(t+1)$	$Q_2(t+1)$	$Q_3(t+1)$	$q_{s1}$	$q_{s2}$	$q_{s3}$
1	0	0	0	0	0	1	0	0	1
1	0	0	1	0	1	0	0	1	1
1	0	1	0	0	1	1	0	0	1
1	0	1	1	1	0	0	1	1	1
1	1	0	0	1	0	1	0	0	1
1	1	0	1	1	1	0	0	1	1
1	1	1	0	1	1	1	0	0	1
1	1	1	1	0	0	0	1	1	1



*Binary up-counters* may utilize various forms of carry. For their analysis it is convenient to use a combined state and excitation table such as shown in Table 6.8. The table applies to a three-bit binary up-counter built around  $T$  flip-flops. The previous states of the flip-flops are entered in binary form in columns 2, 3 and 4.

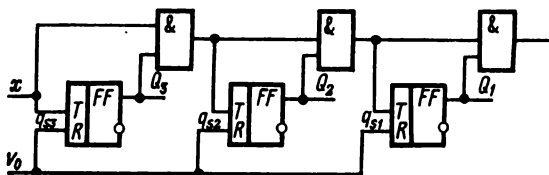


Fig. 6.18

The next states of the flip-flops, also in binary code, are marked in columns 5, 6 and 7. Columns 8, 9 and 10 define the excitation functions of the flip-flops, as found from their excitation matrix, according to the transition required. Referring to Table 6.8, the excitation functions of the counter flip-flops may be written as

$$q_{s3} = x, \quad q_{s2} = xQ_3, \quad q_{s1} = xQ_2Q_3 = q_{s2}Q_2 \quad (6.4)$$

Expression (6.4) leads to at least three logical designs of a binary up-counter, differing in speed of operation. Figure 6.18

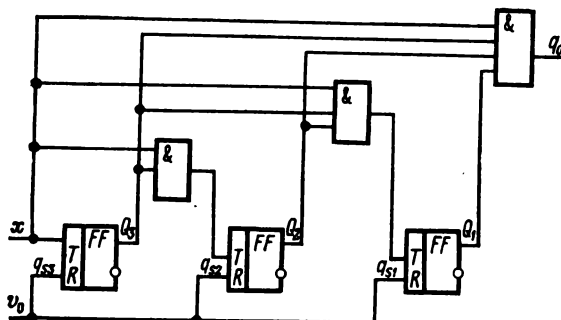


Fig. 6.19

shows a binary counter with the ripple-through carry for which the maximum switching time is given by

$$t_{sw} = \tau_{FF} + (n - 1) \tau_{CD}$$

where  $\tau_{FF}$  = time needed for a flip-flop to operate

$\tau_{CD}$  = time needed for the coincidence detector to operate.

$n$  = number of stages (bit positions) in the counter

The counter of Fig. 6.19 uses a block carry, that is, the carry in any  $k$ th bit position is generated independently of that in the preceding bit position so that the carries out of all bits in a block occur at the same time, according to the excitation function of the flip-flop in the  $k$ th bit position

$$q_{sk} = xQ_{k+1}Q_{k+2} \dots Q_{n-1}Q_n$$

With the block carry, the maximum switching time of the counter is shorter than it is in the previous modification and is given by

$$t_{sw} = \tau_{FF} + \tau_{CD}$$

However, block-carry counters need coincidence detectors with a greater number of inputs, which may in some cases be unattractive. This is why use is made in some cases of a combination

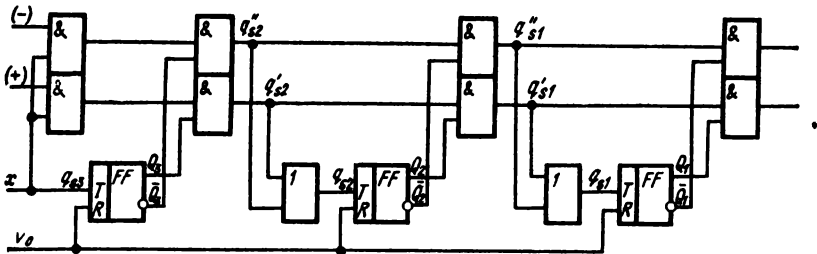


Fig. 6.20

of the ripple-through and block carry. With this arrangement, the counter is divided into blocks, the ripple-through carry being used between the blocks and the block carry within each block.

In both modifications, each flip-flop has a reset input which accepts a reset pulse whose duration should be such that  $\tau_{reset} > t_{sw}$ . The delay lines provided at the inputs to the flip-flops cater for proper timing in transfer of information.

*Reversible (or bidirectional) counters* operate as up-counters when the input is  $y = 0$ , and as down-counters when the input is  $y = 1$  (Fig. 6.20). A combined state and excitation table in binary form for a three-bit reversible binary counter is given in Table 6.9. In this table, the first eight rows ( $y = 0$ ) apply when the counter operates as an up-counter, and they are filled in the same manner as the respective lines in Table 6.8. For  $y = 1$ , when the counter is operating as a down-counter, the binary numbers (representing the new states of the flip-flops) in columns 6, 7 and 8 are unity less than the binary numbers representing their previous states.

Table 6.9

1	2	3	4	5	6	7	8	9	10	11
$x$	$y$	$Q_1(t)$	$Q_2(t)$	$Q_3(t)$	$Q_1(t+1)$	$Q_2(t+1)$	$Q_3(t+1)$	$q_{s1}$	$q_{s2}$	$q_{s3}$
1	0	0	0	0	0	0	1	0	0	1
1	0	0	0	1	0	1	0	0	1	1
1	0	0	1	0	0	1	1	0	0	1
1	0	0	1	1	1	0	0	1	1	1
1	0	1	0	0	1	0	1	0	0	1
1	0	1	0	1	1	1	0	0	1	1
1	0	1	1	0	1	1	1	0	0	1
1	0	1	1	1	0	0	0	1	1	1
1	1	0	0	0	1	1	1	1	1	1
1	1	0	0	1	0	0	0	0	0	1
1	1	0	1	0	0	0	1	0	1	1
1	1	0	1	1	0	1	0	0	0	1
1	1	1	0	0	0	1	1	1	1	1
1	1	1	0	1	1	0	0	0	0	1
1	1	1	1	0	1	0	1	0	1	1
1	1	1	1	1	1	1	0	0	0	1

According to Table 6.9, the excitation functions of the  $T$  flip-flops will be

$$\begin{aligned}
 q_{s3} &= x, & q_{s2} &= xQ_3\bar{y} \vee x\bar{Q}_3y = q'_{s2} \vee q''_{s2} \\
 q_{s1} &= x\bar{y}Q_2Q_3 \vee xy\bar{Q}_2\bar{Q}_3 = q'_{s2}Q_2 \vee q''_{s2}Q'_2 = q'_{s1} \vee q''_{s2} \quad (6.5)
 \end{aligned}$$

The reversible counter which implements the block diagram of Fig. 6.20 uses the ripple-through binary carry according to Eqs. (6.5).

**Scalers** (or scaling circuits) produce an output after a predetermined number of input pulses has been received. The number of input pulses per output pulse is called the scaling factor or the scaling ratio. Incidentally,  $n$ -bit binary counters are all binary scalars for which the scaling factor or ratio is  $k = 2^n$ .

Scalars can be organized in any one of two ways, according to the manner chosen to represent the internal states of the flip-flops.

(1) A zero-initial-state scaler with a factor of  $k > 2$  is arranged so that all states, starting with the zeroth one, are represented by binary numbers in increasing order. When the scaler receives the  $k$ th pulse, the circuit resets, and a control pulse appears at its output. The number,  $n$ , of flip-flops needed to implement a scaling circuit is an integer nearest to  $\log_2 k$ .

In the light of the foregoing, a combined transition and excitation table for a scaler with  $k = 10$  may be presented in binary form as shown in Table 6.10.

Table 6.10

1	2	3	4	5	6	7	8	9	10	11	12	13
$x$	$Q_1(t)$	$Q_2(t)$	$Q_3(t)$	$Q_4(t)$	$Q_1(t+1)$	$Q_2(t+1)$	$Q_3(t+1)$	$Q_4(t+1)$	$q_{s1}$	$q_{s2}$	$q_{s3}$	$q_{s4}$
1	0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	1	0	0	1	0	0	0	1	1
1	0	0	1	0	0	0	1	1	0	0	0	1
1	0	0	1	1	0	1	0	0	0	1	1	1
1	0	1	0	0	0	1	0	1	0	0	0	1
1	0	1	0	1	0	1	1	0	0	0	1	1
1	0	1	1	0	0	1	1	1	0	0	0	1
1	0	1	1	1	1	0	0	0	1	1	1	1
1	1	0	0	0	1	0	0	1	0	0	0	1
1	1	0	0	1	0	0	0	0	1	0	0	1

Written as Boolean (switching) functions, the excitations of the flip-flops entered in Table 6.10 have the form

$$q_{s4} = x, \quad q_{s2} = x\bar{Q}_1Q_4$$

$$q_{s2} = x\bar{Q}_1Q_3Q_4 = q_{s3}Q_3 \quad (6.6)$$

$$q_{s1} = x\bar{Q}_1Q_3Q_4Q_2 \vee xQ_1\bar{Q}_2\bar{Q}_3Q_4 = q_{s2}Q_2 \vee xQ_1\bar{Q}_2\bar{Q}_3Q_4$$

and the respective block diagram appears in Fig. 6.21.

The scaler with  $k = 10$  shown in Fig. 6.21 may be used as a stage for a BCD counter using the 8421 weighted code, with the outputs of the flip-flops  $Q_1$ ,  $Q_2$ ,  $Q_3$  and  $Q_4$  used as the outputs of the overall network.

(2) A nonzero-initial-state scaler has an initial state represented as  $m = 2^n - k$ , while the remaining states are represented by a sequence of binary numbers in ascending order, as in the previous case. Accordingly, in block-diagram terms, such a scaler will be not unlike a binary up-counter in which the carry out of the most significant bit position is coupled by OR gates to the inputs of the flip-flops initially residing in the 1 state. For example, with  $k = 13$ , we have  $n = 4$ , and  $m = 2^n - 13 = 3$ , whence the initial 1 state will be associated with the flip-flops in the two least significant bit positions (0011).

**6.3.3. Adders.** An adder is a component of the arithmetic unit of any digital computer. It generates the algebraic sum of num-

bers. Adders may be serial or parallel, according as the augend and addend bits are available for addition. In the former case, they come bit after bit, starting with the least significant one. During each bit time, an adder has three inputs two of which are an augend bit and an addend bit both belonging to the same bit position, and the third bit is the carry from the previous addition. In the latter case (a parallel adder), all the addend and augend bits are available simultaneously. In parallel addition the sum may be placed in a sum register which is separate from the addend and augend registers (a combining adder), or the sum may replace the augend (an accumulating adder).

Consider the logical design of a simple (single-bit) adder, which is part of any type of adder. It generates the module 2 sum,  $s$ , of

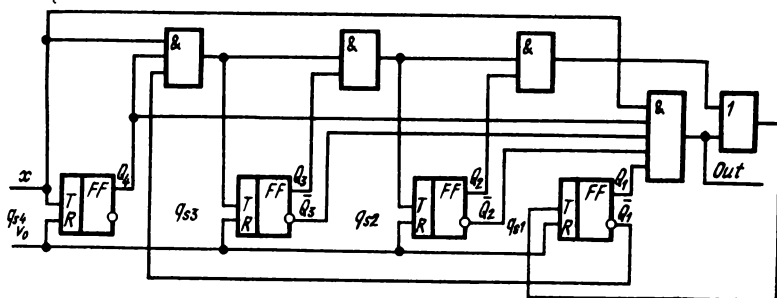


Fig. 6.21

three bits: the augend bit  $x$ , the addend bit  $y$ , and the carry bit  $z$  from the previous stage. This operation of addition can be formalized in terms of Boolean functions as follows:

$$s = x\bar{y}\bar{z} \vee \bar{x}y\bar{z} \vee \bar{x}\bar{y}z \vee xyz \quad (6.7)$$

$$p = \bar{x}yz \vee x\bar{y}z \vee xyz \vee xy\bar{z} \quad (6.8)$$

where  $p$  is the carry into the next stage (bit position).

### Simple Adder Using an Accumulator

In a simple adder using an accumulator, the addend and the augend are available for addition at different times and the sum replaces the augend in the register (or accumulator). The modulo 2 sum of the operands is taken by a  $T$  flip-flop. Since the augend and the addend are available at different times, the expressions for the sum and the carry, (6.7) and (6.8), take the form

$$s = [x(t_1) \oplus y(t_2)] \oplus z(t_3) \quad (6.9)$$

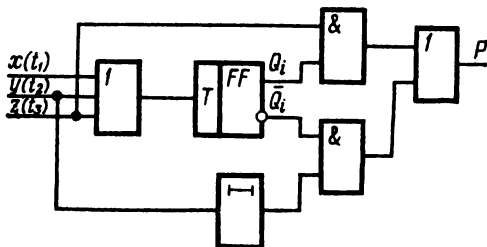
$$p = z(t_3) [x(t_1) \oplus y(t_2)] \vee \overline{[x(t_1) \oplus y(t_2)]} y(t_3) \quad (6.10)$$

where  $t_1$ ,  $t_2$  and  $t_3$  are the instants at which the augend, the addend and the carry are available. The instants  $t_1$  and  $t_2$  must be chosen such that

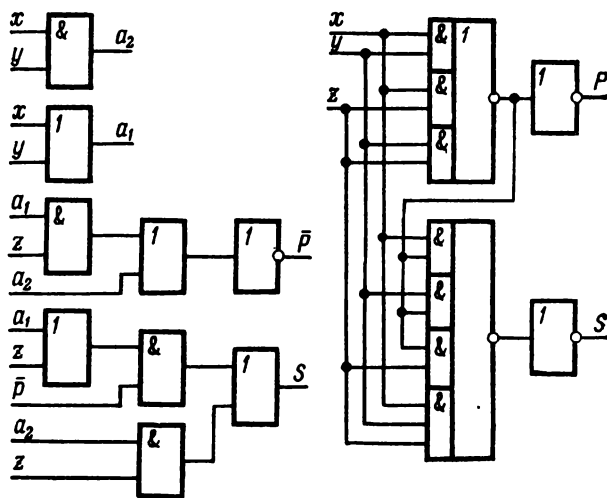
$$t_2 > t_1 + t_0$$

$$t_3 > t_2 + t_0$$

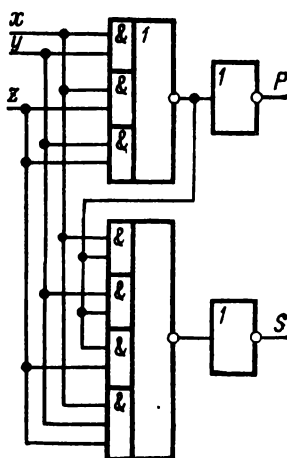
where  $t_0 = \tau_{FF}$ , if the flip-flop is caused to change state by the leading edge of the input pulse, and  $t_0 = \tau_{FF} + \tau_{in}$  if the flip-flop



(a)



(b)



(c)

Fig. 6.22

is caused to change state by the trailing edge of the input signal. Here,  $\tau_{FF}$  and  $\tau_{in}$  are the operate time of the flip-flop and the duration of the input signal, respectively.

The logic diagram of the simple adder incorporating an accumulator and fitting Eqs. (6.9) and (6.10) appears in Fig. 6.22a. The delay line maintains the proper time relationships between the memory elements.

### *Typical Stage of an Adder Using a Separate Sum Register*

With inverting inputs, a single-bit adder using a separate sum register and formalized by the Boolean functions (6.7) and (6.8) will require for its realization four three-input AND gates, three two-input AND gates, one four-input OR gate, and one three-input OR gate.

If a derived function is used to obtain another function as a way of simplifying the given Boolean functions (Sec. 6.2), the circuit implementing it may be arranged to avoid inverse inputs and to reduce the total count of inputs on the logical elements. Thus, after appropriate manipulations, we get

$$\begin{aligned} p &= xy \vee z(x \vee y) = a_2 \vee za_1 \\ s &= \bar{p}(a_1 \vee z) \vee a_2z \\ a_1 &= x \vee y \\ a_2 &= xy \end{aligned} \tag{6.11}$$

The logic diagram fitting Eq. (6.11) appears in Fig. 6.22b. The Boolean expressions for the sum and carry may be recast as follows

$$\begin{aligned} \bar{p} &= \overline{xy \vee xz \vee yz} \\ \bar{s} &= \overline{\bar{p}x \vee \bar{p}y \vee \bar{p}z \vee xyz} \\ p &= \bar{\bar{p}}, \quad s = \bar{\bar{s}} \end{aligned} \tag{6.12}$$

The logic diagram fitting Eqs. (6.12) is shown in Fig. 6.22c.

### *Multibit Adders*

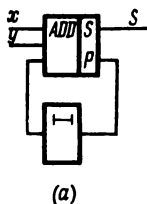
A serial multibit adder uses a single-bit adder with a separate sum register, in which the carry ( $p$ ) output is coupled to the  $z$  input by a delay line which stores the carry for one digit time equal to the repetition period of input signals. (Fig. 6.23a).

A parallel multibit adder is assembled from as many single-bit adders as there are bits in the numbers being added. They may use either a separate sum register and carry propagation or an accumulator and half-sums. The carry may be of the sequential, ripple-through, block or combination (block-and-cascade) type.

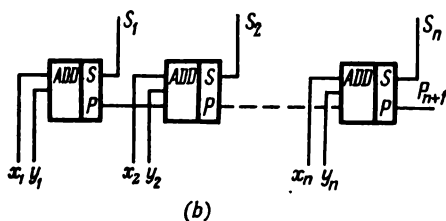
A parallel multibit adder with an accumulator consists of single-bit adders like that shown in Fig. 6.22a. In such adders, the carry ( $p$ ) output from the  $i$ th stage is coupled to the  $z$  input of the

( $i - 1$ )st stage. If an adder operates on numbers in 1's complement representation, the carry out of the most significant bit must be connected to the  $z$  input of the least significant bit (the end-around carry).

Where speed of operation is a major consideration, use may be made of the block-cascade carry along the same lines as in binary counters.



(a)



(b)

Fig. 6.23

Parallel multibit adders with a sum register are made up of single-bit adders whose outputs are connected to the sum register. In arranging for a binary carry in such adders, it must be sought to minimize the number of stages in the carry circuits.

**6.3.4. Decoders.** The function of decoders is to convert a coded word into a representative signal. The operation of decoding is formalized by a system of Boolean (switching) functions each of which has the value 1 for only one set of

variables. As a rule, a decoder accepts signals from a register whose states are to be converted into a control signal. Decoders are widely used in storage and control units.

The system of functions mechanized by a decoder consists in the general case of  $2^n$  functions of the form

$$z_i = x_1^{\sigma_1} x_2^{\sigma_2} \dots x_n^{\sigma_n} \quad (6.13)$$

The network mechanizing a system of functions of the form (6.13) without transformation is called a *matrix decoder*. A matrix decoder for two variables is shown in Fig. 6.24a.

The number of elements needed to implement a decoder may be drastically reduced by appropriately simplifying the original functions of the system. For example, a system of functions of four variables, such as (6.13), may be simplified as follows

$$\begin{aligned} z_0 &= p_0 p_0^1, & z_4 &= p_1 p_0^1, & z_8 &= p_2 p_0^1, & z_{12} &= p_3 p_0^1 \\ z_1 &= p_0 p_1^1, & z_5 &= p_1 p_1^1, & z_9 &= p_2 p_1^1, & z_{13} &= p_3 p_1^1 \\ z_2 &= p_0 p_2^1, & z_6 &= p_1 p_2^1, & z_{10} &= p_2 p_2^1, & z_{11} &= p_3 p_2^1 \\ z_3 &= p_0 p_3^1, & z_7 &= p_1 p_3^1, & z_{11} &= p_2 p_3^1, & z_{15} &= p_3 p_3^1 \end{aligned} \quad (6.14)$$



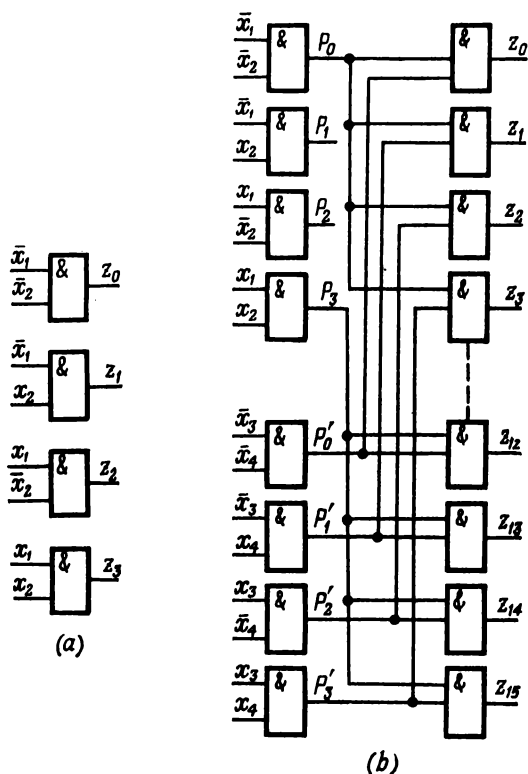


Fig. 6.24

$$\begin{aligned}
 p_0 &= \bar{x}_1 \bar{x}_2, & p'_0 &= \bar{x}_3 \bar{x}_4 \\
 p_1 &= \bar{x}_1 x_2, & p'_1 &= \bar{x}_3 x_4 \\
 p_2 &= x_1 \bar{x}_2, & p'_2 &= x_3 \bar{x}_4 \\
 p_3 &= x_1 x_2, & p'_3 &= x_3 x_4
 \end{aligned} \tag{6.15}$$

A decoder fitting the system of functions represented as (6.14) and (6.15) is called a *square decoder* (Fig. 6.24b). If each function of four variables of the form (6.13) is transformed to

$$z_i = \{[(x_1^{\sigma_1} x_2^{\sigma_2}) x_3^{\sigma_3}] x_4^{\sigma_4}\} \tag{6.16}$$

the realization will be called a *pyramidal decoder*.

## Chapter VII

### Principal Units of an Electronic Digital Computer

#### 7.1. Arithmetic Unit

**7.1.1. Purpose, basic characteristics, classification and elements.** The arithmetic unit, which is one of the five major functional sections of all general-purpose electronic digital computers, provides the facility for performing arithmetic and logical operations.

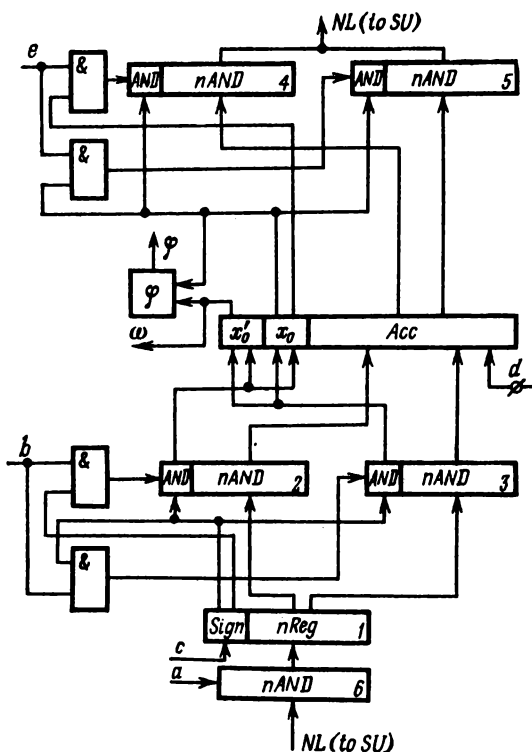
An arithmetic unit may be characterized in terms of the operations it can perform, speed of operation expressed in terms of the execution time, the number system adopted, the range of digits, number representation (fixed or floating point), the manner in which the operations are performed, the accuracy desired, the reliability achieved.

According to the manner in which multi-digit numbers are processed, there may be serial, parallel and serial-parallel arithmetic units. In a parallel arithmetic unit, all the digits (operands) are available for processing simultaneously; in a serial arithmetic unit, operands are processed as they appear serially (bit after bit) in time.

According to number presentation, there may be fixed-point and floating-point arithmetic units. According to the number system adopted, arithmetic units may be binary, decimal, etc. Like adders, arithmetic units may use either sum registers and carry propagation or accumulators for their operation.

An arithmetic unit consists essentially of two subunits, one of which effects the arithmetic and logical operations and the other supplies the control function. Floating-point computers usually incorporate arithmetic units with two operational subunits, one to operate on exponents and the other on magnitudes (mantissas). Fixed-point machines usually have an arithmetic unit with one operational subunit.

As has been shown in Sec. 5.2, any operation in the arithmetic unit reduces to a sequence of elementary operations arranged according to an algorithm. These elementary operations, sometimes referred to as microoperations, are transfer of an operand (addend, multiplier, dividend) to the adder, transfer of an operand from the internal memory to the register, left or right shift of an operand in the register or adder, etc. As a rule, a microoperation is executable within a single clock time and is initiated by a suitable pulse supplied by the control unit of the arithmetic section.



**Fig. 7.1.**

**7.1.2. Logical design of an arithmetic section for the operations of addition and subtraction.** As an example of logical design, consider an arithmetic unit operating on fixed-point binary numbers stored in sign-and-magnitude form. The arithmetic unit is to generate an overflow signal,  $\phi$ , and a negative-result signal,  $\omega$ . It will use a parallel adder with an accumulator.

According to the algorithm of the desired operations (see Sec. 5.2) and the basic requirements for the arithmetic section, its operational unit should include the following essential assemblies (Fig. 7.14):

- (1) A parallel adder with an accumulator and an end-around carry, *PA*. The sign-bit input lines of the adder are tied together.
- (2) A register, *nReg-1*, to store the number transferred from the memory section; the sign-bit stage uses an *RST* flip-flop. The *T*

input (line *c*) may accept a signal to change the sign of the number stored in the register.

(3) AND gates,  $n\text{AND-6}$ , to read numbers from the memory section into  $n\text{Reg-1}$ .

(4) AND gates,  $n\text{AND-2}$  and  $n\text{AND-3}$ , to transfer numbers from  $n\text{Reg-1}$  to the adder in sign-and-magnitude or 1's complement form, respectively.

(5) AND gates,  $n\text{AND-4}$  and  $n\text{AND-5}$ , to transfer the result in sign-and-magnitude representation from the adder to the memory section.

(6) A circuit to generate the overflow signal,  $\phi$ .

(7) Two AND gates to control the transfer of a number from the adder.

(8) Two AND gates to control the transfer of a number from  $n\text{Reg-1}$ .

In Fig. 7.1, the input lines are labelled with lower-case letters from the beginning of the alphabet. The signals appearing on the control lines will be designated as  $v_i$ , where *i* is the subscript of the respective control line.

To carry out the operation of addition, with the augend and the addend stored in the memory section, the control unit of the arithmetic section must generate the following sequence of signals on the control lines:  $v_d, v_a, v_b, v_a, v_b, v_e$ , where  $v_d$  is the pulse that clears the adder to zero. To carry out the operation of subtraction, the control unit of the arithmetic section must generate control signals in the following sequence:  $v_d, v_a, v_b, v_a, v_c, v_b, v_a$ .

**7.1.3. Logical design of an arithmetic section for the operations of multiplication.** In the logical design of an arithmetic unit for the operation of multiplication, we shall assume an arithmetic section having the same performance as in the previous case with the addition of two more requirements, namely, that the multiplicand and the multiplier will be in sign-and-magnitude form and that round-off will be applied. The algorithm to be realized is Mode 1 in Table 5.2. In this mode, the multiplicand is stationary, while the multiplier and the sum of partial products are shifted to the right, that is, multiplication proceeds, starting with the least significant bit of the multiplier.

To satisfy this algorithm and the basic requirements for performance, the operational unit of the arithmetic section must include the following assemblies (Fig. 7.2):

(1) A parallel adder with an accumulator, with capabilities for right shift, but with no shift in the sign adder.

(2) A register,  $(n - 1) \text{ Reg-1}$ , to store the magnitude of the multiplicand.

(3) A shift register,  $(n - 1) \text{ Reg-2}$ , to store the magnitude of the multiplier.

(4) AND gates,  $n$ AND-1 and  $n$ AND-2, to steer the multiplicand and multiplier to the respective registers, and also to transfer the sign bits of the multiplicand and multiplier to the sign adder.

(5) AND gates,  $(n-1)$  AND-3 and  $(n-1)$  AND-4, to transfer the magnitude of the multiplicand to the adder and the result to the memory section, respectively.

(6) AND gates, AND-1 and AND-2, to control transfer of the multiplicand to the adder and to carry out the round-off operation in the  $(n+1)$ st bit position, respectively.

To carry out the operation of multiplication in accordance with the chosen algorithm, the control unit of the arithmetic section

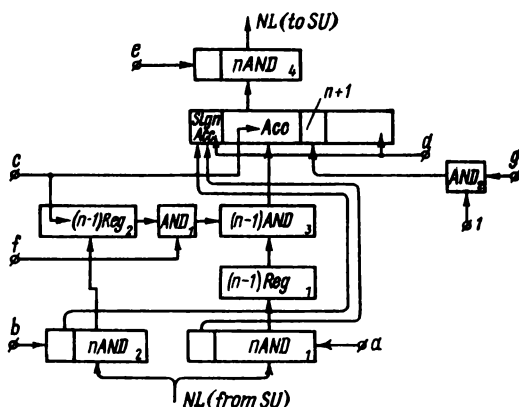


Fig. 7.2

must generate the following sequence of control signals:  $v_d$ ,  $v_a$ ,  $v_b$ ,  $[v_f, v_c]_n$  times,  $v_g$ ,  $v_e$ . For this algorithm of multiplication, where the multiplier and the sum of partial products are shifted the same number of places, the least significant bit stages of the adder may be utilized as a multiplier register, which makes a separate multiplier register unnecessary.

**7.1.4. Logical design of an arithmetic section for the operation of division.** Basically, the performance criteria of an arithmetic section in the case of division are the same as for an arithmetic section to carry out addition and subtraction, with the added requirement that the operation of subtraction in an adder is carried out by placing the subtrahend in the adder in its 2's complement form. The operation of division will be performed according to Mode 2 of the division algorithm presented in Sec. 5.2, with the dividend arranged to arrive in the arithmetic section before the divisor.

To satisfy the chosen algorithm and the basic performance requirements, the operational unit should include the following assemblies (Fig. 7.3):

- (1) A parallel adder with an accumulator, without end-around carry. In the adder, the sign will be presented by two bits,  $x_0'$  and  $x_0$ .
- (2) A divisor register,  $nReg-1$ , to store the divisor.
- (3) A divisor 2's complemener.
- (4) A quotient shift register,  $nReg-2$ .
- (5) AND gates,  $nAND-2$ , to transfer the dividend to the adder and the divisor in sign-and-magnitude form to the adder if the remainder is negative,  $\omega = 1$ .

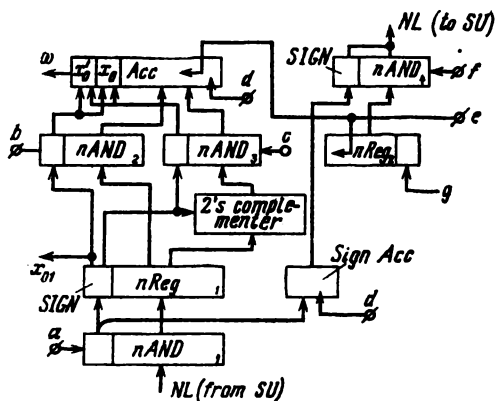


Fig. 7.3

- (6) AND gates,  $nAND-3$ , to transfer the divisor in 2's complement form to the adder if the remainder is positive,  $\omega = 0$ .
- (7) AND gates,  $nAND-4$ , to transfer the quotient to the memory section.
- (8) A quotient sign adder to obtain the sign of the result.

To carry out the operation of division according to the chosen algorithm, the control unit of the arithmetic section should generate control signals in the following sequence:  $v_d$ ,  $v_a$ ,  $v_b$ ,  $v_a$ ,  $[v_e, (v_b\omega \vee v_e\bar{\omega})]$ ,  $v_g$   $n$  times,  $v_f$ .

## 7.2. Storage

**7.2.1. Classification and basic characteristics of storage units.** The storage (or memory) unit of a general-purpose digital computer stores both the instructions that control machine operations and the data that are to be processed, and supplies them to other computer sections.

Digital computers use storage units widely differing in performance (notably, capacity), operating principle, and physical realization. In fact, a single computer may use a hierarchy of different storage units, each performing a specific function.

Basically, storage units may be classed as follows:

(1) A storage unit which is an integral part of a computer and performs its designated functions in the course of operations carried out by the arithmetic and control units is usually called an internal storage (or memory). If data are entered once for all and can be read out repeatedly we shall have a fixed or read-only storage. A synchronizing or time-matching storage unit, that is, one holding information while it is being transferred between the computer sections operating at different speeds, will be called a buffer storage. A storage divorced from the computer proper and designed for bulk storage and exchange of information with the internal and buffer storage units will be referred to as external storage.

(2) Another way of classifying storage units is according to the manner in which data can be entered (stored) or retrieved. We shall call a unit a random access storage if data can be entered in and retrieved from it in any arbitrary sequence. In contrast, if data can be entered in or retrieved from storage in a pre-arranged address sequence, we shall have a sequential access storage. Where a storage system references storage locations by data content, we shall call it an associative storage. Finally, there may be an address-organized storage in which a particular storage location is found by an exhaustive search from address to address.

(3) Storage may be static or dynamic. With static storage, the device stores data so that they do not change position. With dynamic storage, the device stores data so that they can move or vary with time, and the specified data are not always available for retrieval.

(4) If the digits of a selected word are available from a storage unit all at the same time, we shall call it a storage with parallel representation. In contrast, where the digits of a word are available one after another at the same terminal, we shall call it a storage unit with serial representation. Obviously, a storage unit with serial-parallel representation may be, and is, used.

(5) A very important feature of any storage is whether or not information is destroyed during readout. Accordingly, we have destructive readout, where storage must incorporate means for regenerating the information lost, and nondestructive readout where the information remains intact.

(6) A variety of materials, or storage media may be utilized for storage, such as magnetic cores with a square hysteresis loop, semiconductor devices, etc.

(7) Finally, storage units may differ in mechanical construction, methods used in production, etc.

The most important performance characteristics of a storage unit are speed, capacity and reliability. The speed of a storage unit has a direct bearing on the operating speed of the entire computer because problem-solving involves a huge number of data and instruction transfers between storage and the rest of the computer. For the internal, buffer and fixed storage, it is customary to measure the memory *cycle time*,  $T_c$ , which is the minimum time from the beginning of one access to the beginning of the next. Its reciprocal, called the maximum cycle rate of a storage,  $F_{\max}$ , is also an important speed characteristic.

For internal and fixed (or read-only) memories it is customary to specify their access time which is defined as the time from the instant an address has been transferred to the storage and the instant when the word (or number) is available at its output terminal, and is symbolized as  $T_a$ . Both the cycle time and the access time of a storage are expressed in microseconds,  $\mu s$ . The cycle time is generally longer than the access time, that is,  $T_a < T_c$ .

For external memory, the speed is defined in a somewhat different manner. The reason for this difference is that external memory handles information in blocks rather than in individual numbers (or words), as happens in internal memory. Hence, the speed of external memory is the sum of its maximum storage and retrieval times,  $T_{\max}$ , for a block of numbers (or words). Practically,  $T_{\max}$  may turn out to be shorter than  $T_c$ . Another speed characteristic for external memory is the rate,  $F$ , at which words or numbers in parallel representation can be stored into or retrieved from a block.

Storage capacity is generally measured by the number of machine words of a stated base or the equivalent number of binary digits (bits) that a storage can hold at a time.

The reliability of a storage unit must be such that the associated computer can solve problems under the operating conditions stated in the specifications. Accordingly, the reliability of storage units may be stated in terms of failure rate (the number of failures per unit time or the number of valid signals per invalid signal), or as the range of operating conditions, or both.

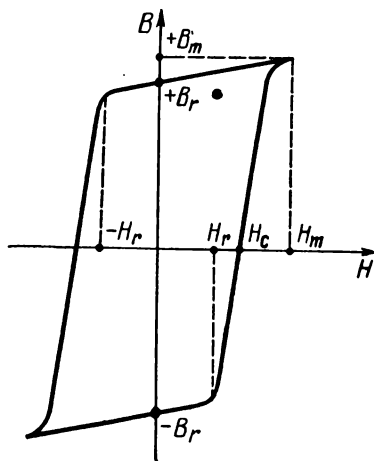
The performance characteristics of storage units are closely interrelated, and an attempt to improve one may impair some other. For example, an increase in capacity in internal storage inevitably leads to a decrease in its speed and reliability.



There are also economic characteristics of storage units, such as cost of storage per bit stored, mechanical characteristics such as number of bits stored per unit volume. As regards external memories, account is taken of writing density,  $\rho$ , which shows how many units of information can be packed into unit length of storage medium.

**7.2.2. Storage elements and storage media.** A physical device intended to store one bit of information is called a *storage element*. This may be a material or a device having two stable states readily recognizable and controllable.

The most commonly used type is *magnetic-core* storage elements with a rectangular hysteresis loop. They may reside in two stable states of magnetization, each characterized by a residual (or remanent) flux density of its own. A typical steady-state  $B$  (flux density) versus  $H$  (field strength) characteristic is shown in Fig. 7.4. It is usual to designate the positive residual state,  $+B_r$ , as binary 0, and the negative residual state,  $-B_r$ , as binary 1\*. After a magnetic core has been set in this binary state, it will remain there for any length of time. These states are recognized (sensed) by applying current pulses which produce magnetizing force,  $+H_m$ .



Eig. 7.4

If a core has previously been set to the binary 0 ( $+B_r$ ) state, application of  $+H_m$  will in fact be in the same sense, and a relatively little flux change will occur, e.g., from  $+B_r$  to  $+B_m$  with the rising current, and back from  $+B_m$  to  $+B_r$  with the falling current. If, on the other hand, the core has previously been set to the 1 ( $-B_r$ ) state,  $+H_m$  will pulse it in the opposite sense (causing a change of state), and the flux density will change by a considerable amount, e.g., from  $-B_r$  to  $+B_m$  with the rising current and back from  $+B_m$  to  $+B_r$  with the falling current. This change in flux density (or magnetic flux  $\Phi = SB$ , or flux linkage  $\Psi = \omega\Phi$ ) induces an emf in the sense wire, this emf being the greater, the greater the change in flux density and the faster the rate of change, that is,  $e \propto dB/dt \propto \Delta B/\Delta t$ .

\* As often, the reverse may be true. — Tr.

As is seen, both the change in flux density and the rate of change depend on the residual state ("0" or "1") in which the core has been set. Therefore, the emf induced in the sense wire will have amplitude, duration and waveform wholly governed by the binary state of the core, and this forms the basis of recognizing (or sensing) the state written into the core.

A magnetic core is switched between residual states by passing a pulse of current through it, so as to produce a magnetic field of intensity  $\pm H$ . During a read (sensing) operation, the core switches to the residual state representing the binary 0; during a

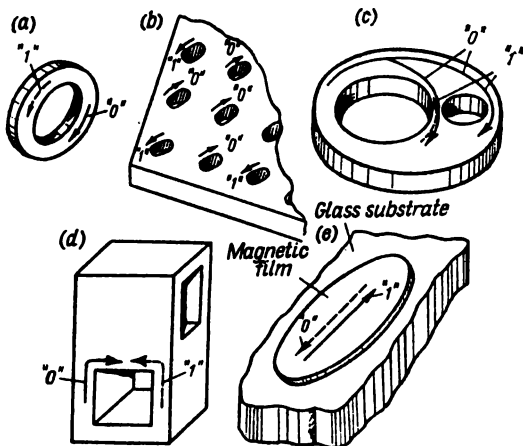


Fig. 7.5

write operation it will switch to the residual state representing the binary 1 if the applied magnetizing force is  $|-H| \geq |-H_m|$ , or will remain in the 0 state if  $|-H| \leq |-H_r|$ . This read operation destroys the information stored.

Of all forms of magnetic cores used in internal and buffer storage, the one most commonly used is the moulded-ferrite type pressed into toroid (ring or doughnut) shapes (Fig. 7.5a). Ferrite cores are available in a range of outside diameters from 0.5 to 2 mm, and in a variety of source materials, temperature characteristics, hysteresis loop shape, etc. The directions of residual magnetization are shown by the arrows in Fig. 7.5a, each of which may be selected to represent the binary 1 or the binary 0. Each core is strung on wires that control its residual state and one more wire that senses this state. Unfortunately, the read operation destroys the stored information.

The handling and fabrication of individual cores may be eliminated through the use of *multiaperture ferrite plates*, such as

shown in Fig. 7.5*b*. The areas around the holes operate in the same manner as individual ferrite cores. The arrows in the figure show the directions of residual magnetization for each hole.

Although simple to fabricate, storage elements with destructive readout involve an additional time required to rewrite (regenerate) the lost information, which fact impairs the speed of the storage unit as a whole. There are magnetic storage elements which are free from this limitation. Examples are a secondary-aperture ferrite core like the Transfluxor manufactured by RCA, which has two openings (Fig. 7.5*c*). The arrows in the figure show the directions of the residual flux for binary 0 and 1. Switching between the states is initiated by a current pulse applied to the wire threading the larger hole. To set the core to the 0 state, a large mmf is applied to the core by means of the winding through the large aperture, so that the entire core is saturated in one (say, clockwise) direction. A winding (called a drive, excitation or interrogation winding) through the small aperture can cause no change of flux around that hole because the material is saturated, so that another (sense or output) winding through the small aperture can produce no output when the core is in this state, and the core is said to be *blocked*. In order to write a 1 into the core, an mmf in the reverse direction is applied to the core by means of the winding through the large hole. The mmf must be just sufficient to reverse the field only around the large aperture, without affecting that around the small hole. Now the two fluxes are in opposite directions, and the core is said to be *unblocked*. If an alternating mmf or bipolar pulses are applied by means of the interrogation winding, an output will appear on the sense winding.

Nondestructive readout can also be obtained with devices based on a quadrature (or transverse) field, that is, one at right angles to the residual flux. Within certain limits, the magnetization caused by the quadrature field in the material is a linear function of the field strength, has the same polarity, and falls to zero when the quadrature field is removed.

An example of a storage element based on a quadrature field is a special ferrite core with two apertures in orthogonal directions (a *biaxial ferrite core*), such as shown in Fig. 7.5*d* which also gives the directions of the residual flux (saturation) for the 0 and 1 states. The core is set to one of these states by a pulsed field applied by means of a wire or winding through the lower aperture. The material around the lower aperture has a magnetization curve similar to that shown in Fig. 7.4. The top aperture is used to produce a pulsed quadrature field by means of a wire through that aperture for readout. The quadrature field causes a change in the horizontal component of the original field, so that

a pulse of emf is induced in the sense (output) winding, of a polarity determined by the residual flux, that is, the information, stored. In this way, the sign of the output pulse uniquely defines the stored bit as a 0 or as a 1.

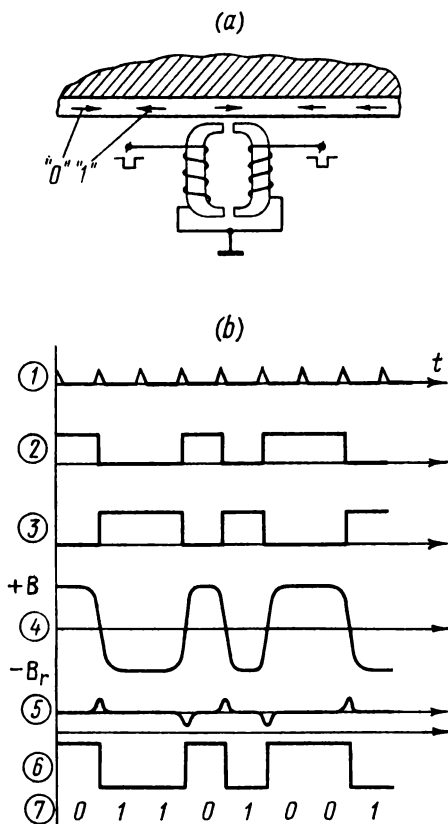


Fig. 7.6

Storage cells can be produced by depositing rectangular patches of *thin magnetic film* on a suitable substrate. This film is anisotropic, that is, when placed in a strong magnetic field, it exhibits an easy axis of magnetization in the direction of the field. Its hysteresis loop is similar to that shown in Fig. 7.4, and so thin-film patches may be set to the binary 0 and 1 states as shown by the arrows in Fig. 7.5e. The magnetic field used for this purpose must have a magnitude and a polarity to suit the magnetization curve of the thin film. The patches are interrogated with a quadrature field, as in the case of biaxial-aperture ferrite cores. The control and interrogation fields are applied by means of current pulses in appropriate wires which are all made (or, rather, printed) as metal films.

For example, the interrogation wire is printed on top of the thin-film patches in the direction of the residual flux. Then a current pulse in the wire will set up a field at right angles to the direction of saturation. Thin-film storage elements are extremely small and can conveniently be fabricated in large numbers in a single operation.

External memory ordinarily uses storage media in the form of a ferromagnetic material deposited on a suitable nonmagnetic substrate (magnetic drums, magnetic tape, or magnetic discs). This ferromagnetic material has a hysteresis loop approaching that shown in Fig. 7.4, which fact indicates the possibility of driving the material to two states of residual magnetization. As

has been shown, materials with this property can be used for binary storage. For the purpose of recording, the magnetic medium (drum, tape or disc) is passed by a magnetic head such as shown in sketch form in Fig. 7.6a. The head produces an intense localized magnetic field at the surface of the recording medium whose direction and, as a consequence, that of the residual magnetization depends on the signal energizing the coil of the head core. As a result, each succeeding area on the tape (drum or disc) is brought into a magnetized state representing a binary 1 or a binary 0. Thus, information written in this way is stored as a pattern of areas differing in residual magnetization. For read-back the storage medium is again passed by the same or another magnetic head. The external magnetic field along the surface, resulting from the residual magnetization, couples magnetically with the head and induces an emf in the coil, representative of the signal read back. These events are illustrated in the timing diagram of Fig. 7.6b. In writing a binary data item (7), the left-hand and right-hand coils of the magnetic head are energized by current pulses shown at 2 and 3, respectively. The current in the head is switched by a gating signal (6) from a data flip-flop. After the write operation, the residual flux density in the magnetic layer has the form shown at 4. During the read operation, the magnetic flux in the head core, following the variations in the residual magnetization (4) along the surface, induces an emf (5) in the head coil. This output emf is processed into a form suitable to control the state of the data flip-flop and applied to its  $T$  input so that the flip-flop moves through the same sequence of states (6) as in writing the original binary data item (7). Synchronizing pulses (1) provide a means for measuring the duration of the binary states.

High-speed storage elements can be built with tunnel diodes connected into bistable circuits (Fig. 7.7). For its operation, this type of storage depends on the fact that a tunnel diode,  $TD$ , can reside in two stable states, given a positive write voltage,  $V_w$ , a zero read voltage,  $V_r$ , and a negative regenerating voltage,  $V_{reg}$  (see Fig. 7.7a). The slope of the load line is determined by the value of  $R$ , and the diode should therefore have two stable states (see Fig. 7.7b). The low level,  $V_{min}$ , represents a logical 1, and the high level,  $V_{max}$ , a logical 0.

Readout is initiated by applying a negative voltage pulse,  $V_r$ . If the tunnel diode has been set to the 1 state, a negative pulse will pass through the diode  $D$ , of a duration determined by the turn-off time of the tunnel diode. Since the turn-off time is finite the cathode of the tunnel diode will remain at a negative potential for some time, and the diode  $D$  will be conducting. If the tunnel diode has been set to the 0 state, the cathode of the tunnel

diode will be at a positive potential all the time, the diode  $D$  will be in the OFF state, and no pulse will be able to get through.

For a binary data item to be written into the storage element, the voltage  $V_r$  should be negative, and the polarity of  $V_w$  should be reversed. Then the tunnel diode will be switched to the 1 state if the diode  $D$  is OFF, and to the 0 state, if the diode  $D$  is ON. When the tunnel diode is switched to the 1 state,  $V_{reg}$  is incremented; in the 0 state of the tunnel diode, it remains unchanged,

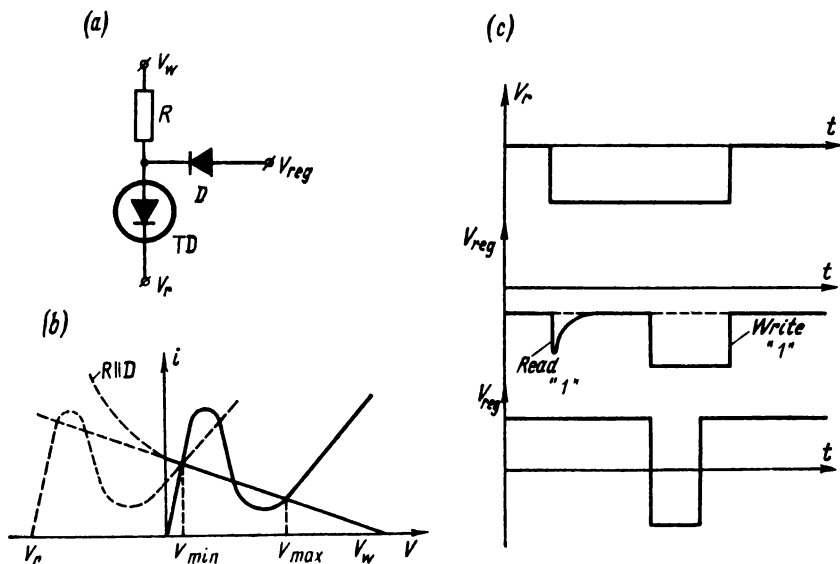


Fig. 7.7

and the tunnel diode goes high. In this condition, the slope of the load line is determined by the parallel connected resistors and the resistance of the conducting diode  $D$ . The timing diagram of the tunnel diode appears in Fig. 7.7c.

Storage elements in which binary states are represented by a weaving pattern are used in fixed (read-only) memories along with storage elements like biaxial ferrites and the Transfluxor in which the states are switched electrically. The manner in which binary states are identified by a fixed weaving pattern in a read-only memory using transformers with toroid cores having a non-rectangular hysteresis loop is illustrated in Fig. 7.8. A wire threading the transformer core represents a binary 1, and a wire threading outside the core represents a binary 0. For readout, a current pulse excitation is applied to one of the threading wires. An

emf will be induced in the output (sense) winding of a core only if the selected wire threads the core.

In dynamic memories, information storage is based on the fact that signals in a conducting medium (say, electrical pulses in a cable) are propagated at a finite speed, as shown diagrammatically in Fig. 7.9a. Knowing the length of the cable,  $l$ , and the ve-

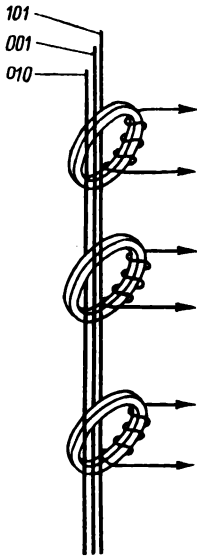


Fig. 7.8

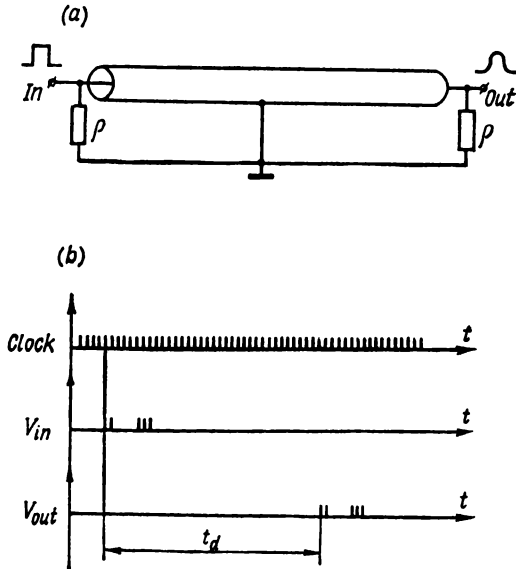


Fig. 7.9

locity at which electric energy is propagated in the cable, it is an easy matter to find the time during which the signal will exist in the cable

$$t_d = l/u$$

Quite appropriately, this type of storage is called a delay-line memory. The synchronizing, input and output signals existing in a delay-line memory are shown in the timing diagram of Fig. 7.9b. Information is stored as a sequence of bits, pulses representing 1's and the absence of pulses representing 0's, and these two states are sensed at the input or output of the delay-line memory at the instants when synchronizing (or clock) pulses occur. Information is stored for the duration of the time delay,  $t_d$ . In the passage through a delay line, however, the signal loses much of its energy and of its true shape. This is why pulse regeneration has to be employed in delay-line memories: the pulses coming out of the delay line are amplified, re-shaped and fed back into the

delay-line input. Stored information is thus circulated indefinitely, and readout takes place at the time of pulse regeneration as output from the reshaping circuit. Data items are written into a delay-line memory by applying appropriate pulses, synchronous with clock pulses, to the delay-line input. The capacity of a delay-line memory in terms of bits simultaneously held in the delay line cannot exceed the number of clock-pulse periods that make up the time delay of the cable. The delay media ordinarily used in delay-line memories may be special alloys, mercury or artificial electrical delay lines.

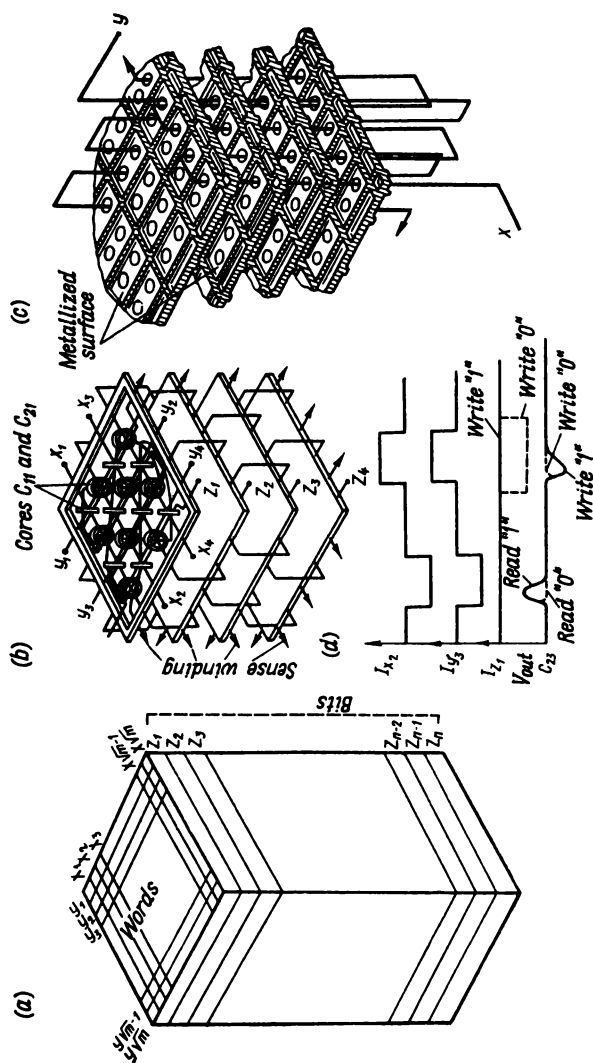
The general trend in the development of storage elements has of late been towards reduced size, cost and drive power, improved reliability and environmental stability (towards temperature, moisture, etc.), and automatic fabrication. Special promise in this respect is held out by thin films, ferrite microcores, multi-aperture devices (*MAD*), and ferroelectric memory elements.

**7.2.3. Organization of storage elements in the internal storage.** In computer memory information is stored in discrete form known as words. Each word is loaded into and retrieved from a separate storage location, or *cell*, which is assigned an *address*. These storage locations are organized to ensure the highest speed in transferring data to and from the arithmetic unit, to reduce the count of equipment used, and to improve reliability. This can be done in several ways.

Most often, use is made of the three-coordinate system. Such magnetic-core storage units are alternatively called *matrix*, selection-plane, or coincident-current storage units. In sketch form, they are depicted as cubes (Fig. 7.10a). Physically, a three-coordinate magnetic-core storage unit is an assemblage of matrices or planes each of which mounts the cores storing one bit of all words. Figure 7.10a shows an *X-Y* plane or matrix. As a rule, square planes or matrices are used, and the number of words they contain is given by the product of the *X* rows by the *Y* columns. The various bits of each word are arranged along the *Z*-coordinate. For the magnetic-core array of Fig. 7.10, the number of words is  $m$  and the number of bits in each word (word size) is  $n$ .

The three-coordinate system needs a smaller count of electronic equipment to select the word at a specified address, because each matrix combines the functions of word selection and storage. Refer to Fig. 7.10b which shows a three-coordinate magnetic-core array (where  $m = 16$  and  $n = 4$ ). In order to select a stored word or to write a new word at a specified address, a sequence of two current pulses, called *half-currents*, must be applied to one *X* wire and one *Y* wire. The application of a half-current to any one coordinate, *X* or *Y*, results in the half excitation of a selection





**Fig. 7.10**

plane through the array, insufficient to switch any core in that plane to the opposite remanent state. The same is true of the half-current applied to the other coordinate,  $Y$  or  $X$ . The existence of both half-currents, however, results in full-current excitation at the intersection of the selection planes, and the core at that intersection is switched as appropriate. When a 0 is to be written, a half-current is applied to the  $Z$  wire in the opposite polarity in the  $X$  and  $Y$  wires.

For reliable operation of a coincident-current memory using ferrite cores with a hysteresis loop like that shown in Fig. 7.4, the field intensity due to a half-current must satisfy the following constraint inequalities:

$$H < H_t, \quad 2H > H_t$$

or

$$H_t > H > H_t/2$$

From the last inequality it is seen that a wider hysteresis loop will tolerate more variation in the half-current. A faster switching action needs a stronger field. To meet this requirement and also to improve reliability, coincident-current magnetic-core memories use cores with a high coercive force.

Coincident-current memories using multiaperture ferrite plates do not differ in the operating principle from similar units using toroidal ferrite cores. The differences that exist in their construction are due to the fact that the plates have a multiplicity of apertures, and also to the presence of an electrochemically printed wire used for control along the  $Z$  coordinate and as a sense wire (Fig. 7.10c). The fact that ferrite plates have printed wires is a considerable advantage because it simplifies the assembly of plates into a memory array. Printed wires can safely be applied because ferrite is actually a dielectric (its resistivity is about  $10^8$  ohms/cm). The timing diagram of Fig. 7.10d applies to memories with ferrite plates as well as with ferrite cores.

Another form of organization intended for internal, buffer and fixed memories is the *word-organized* system, also known as the linear or word selection system. It may use two or three coordinates. By tradition, the locations intended to store a word are arranged on the same axis designated  $Z$  for both three- and two-coordinate systems. A word-organized memory unit is shown in Fig. 7.11a and b. The total count of words (numbers) that a word-organized memory can hold is equal to the last subscript of the  $Z$  coordinate. In comparison with a coincident-current memory, a word-organized one needs a greater count of equipment for word selection, because the elements only perform the function of storage. However, with this organization, the requirements are less stringent as regards spread between elements and their sta-

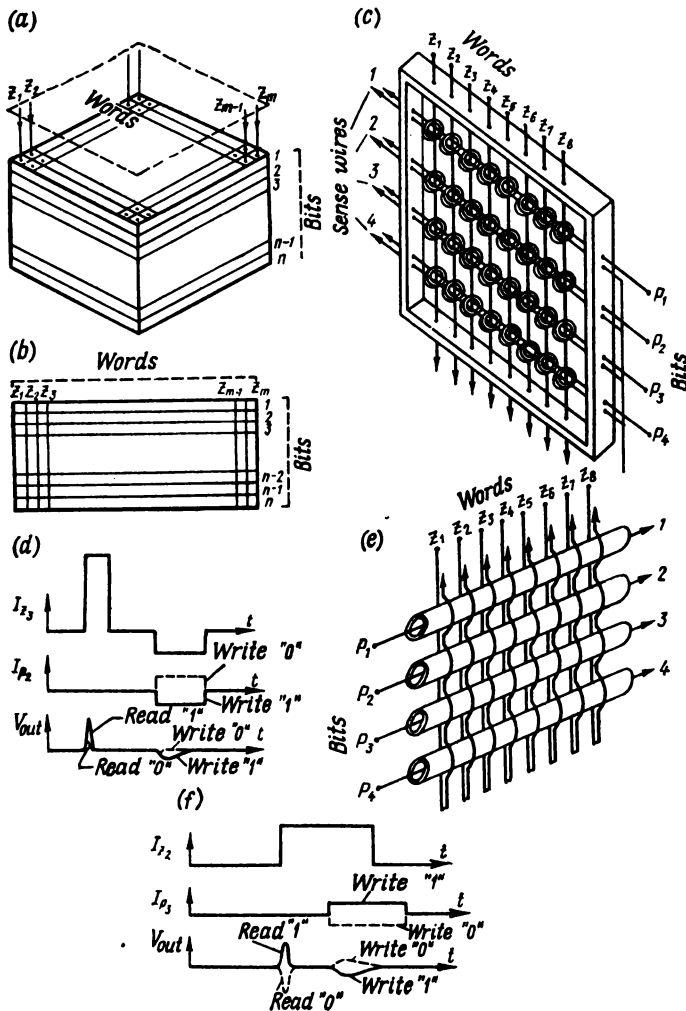


Fig. 7.11

bility, and also (if square-loop ferromagnetic materials are used) faster speed and stronger output signals are obtained because very large drive pulses are applied in reading.

The construction and operation of a word-organized memory unit (with  $m = 8$  and  $n = 4$ ) will be clear from reference to Fig. 7.11c. The drive current in reading a 1 may be arbitrarily large because it acts only on the cores of the selected word. When

a data item is written into a core (Fig. 7.11d), the total field intensity must be sufficient to turn over the cores in writing a 1 and insufficient to do so in writing a 0. The bit write current must be low enough not to switch the cores in the same bit position of unselected words. These requirements are easily met if the cores have a sufficiently square  $BH$  loop (see Fig. 7.4), if the spread in characteristics between the cores is negligible, and if the current amplitudes during a write operation and, as a consequence, the respective fields are such that the following constraints are satisfied:

$$H_b < H_t, \quad H_z + H_b > H_t, \quad H_z - H_b < H_t$$

As is seen, the cycle time of a word-organized memory is reduced solely through a cut-down in the switching time during a read operation, but not in writing.

The bit write wire is also used as a sense wire because the output signal induced in it occurs at a different time from the write signal.

Figure 7.11e shows a sketch of a storage unit with nondestructive readout, which uses glass tubes coated with thin magnetic film (about 10,000 Å thin) as storage elements. The residual flux on these elements is directed around the circumference of the tubes. The timing diagram during a write operation is shown in Fig. 7.11f. During a read operation, no pulses are applied to the bit write wires, and the 0 and 1 output signals differ both quantitatively and qualitatively, in polarity which is decided by the direction of the remanent magnetization (which may be clockwise or anti-clockwise) on the surface of the storage element at the  $Z$  wire. The elements are switched between residual magnetization states by the magnetic field around the tube circumference. The field is set up by the current pulse which exists in the bit wire at the instant when the magnetic material is excited by the quadrature field due to the current pulse traversing the selected wire.

In external storage, information can be stored on the surface of a tape, drum or disc coated with a magnetic material. In contrast to internal or buffer memory, information on the magnetic surface is arranged in a series of areas. Within an area, word bits can be stored or retrieved in parallel, serially, or parallel-serially, depending on the design of the device. Fig. 7.12 shows a parallel magnetic drum on which all bits of a word are stored or retrieved in parallel. The number of words per area and the number of areas per drum depends on the capacity and application of the device. Areas are counted in the direction of drum rotation, each area being identified by a start (SOA) mark. The first area also has a START OF REVOLUTION (SOR) mark. The position of a word along the axis of the drum is located by

synchronizing signals. The number of words an area can hold is less than that of synchronizing marks between two adjacent SOA marks. The synchronizing, SOA and SOR marks improve the reliability with which read and write operations can be controlled. These marks can be applied to the drum surface either electrically before writing information, or mechanically during manufacture, as grooves filled with a magnetic material to produce sudden changes in the magnetic flux readily recognizable during data retrieval.

The magnetic heads used on magnetic drums and discs are often arranged to float on a cushion of air ("air-floated") in order to raise the amplitude of the output signal and to increase the write density. The air gap between an air-floated head and the rotating drum or disc is maintained automatically at a few microns owing to the elastic properties of the entrained air.

A typical storage unit incorporates signal shaping circuits and amplifiers to switch the storage elements and to give standard form to output signals. Switching signals are supplied by drivers which furnish pulses of the desired waveform, amplitude and power. The number of drivers which excite the storage elements of a selected word during a read or write operation can be kept to a reasonable minimum through the use of *switches*. Operation of a half-current switch in the *X*-coordinate of a matrix core storage will be clear from Fig. 7.13. A read or a write signal causes the driver (*DR*) to generate half-currents which are steered by transistors, *T*, into one of the four *X*-lines. In reading, one "row" gate and one "column" gate open simultaneously; in writing the same happens to two other gates corresponding to the selected coordinate. As a result a bidirectional sequence of half-currents is allowed to traverse the selected *X*-line only. When data are retrieved from storage, signals appear at the output, representing the state of the storage elements.

As already noted, these output signals have to be converted to standard form. The need for such conversion will be clear from examination of their waveforms shown in Fig. 7.14. Practical storage elements are not ideal in their behaviour, nor is it possible to make them all perfectly identical. As a result, the signals

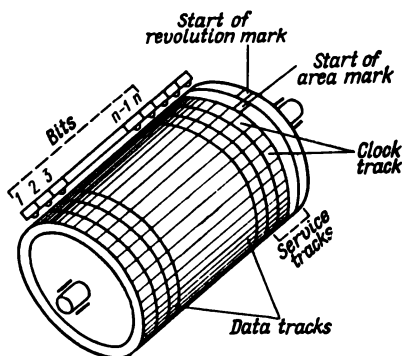


Fig. 7.12



to be stored or retrieved, a function signal to select between a write or a read operation, and a start signal which "tells" the storage control to initiate reading or writing, whichever may be the case. Sometimes, a combined signal is used to select between read and write and to initiate the operation selected. Given these signals, the storage control steers the memory unit as told and generates signals needed to write or read data and separate valid signals from noise. Obviously, the storage control has a direct bearing on storage performance — speed, capacity, cost and size. Whatever storage elements are used (toroidal cores, ferrite plates, quadrature-field devices, etc.), the general principles underlying storage control remain the same, although the type of storage elements will inevitably affect specific requirements for control-signal characteristics.

#### *Control of ferrite-core internal memory.*

As an example, we shall discuss a three-coordinate digit-selection core-matrix memory (Fig. 7.16). The memory cells intended to store  $n$ -bit binary numbers are arranged in a memory module, *MM*. A cell where

a data item is to be stored or retrieved is identified by a binary address code coming over the address line, *AL*, to the memory address register, *MAR*. If the capacity of the memory unit is  $m$  numbers (words), the size of (the number of bits in) an address and, as a consequence, the count,  $n_A$ , of flip-flops in the memory address register will be

$$n_A = \log_2 m$$

Before an address is transferred to it, the memory address register is cleared by a pulse coming from a timing generator, *TG*. From the memory address register, *MAR*, the binary address code is applied to the *X*-decoder, *XD*, and the *Y*-decoder, *YD*, which select appropriate gates in the *X* and *Y* switches *SW*. After a time interval determined by the delays inherent in the decoders and

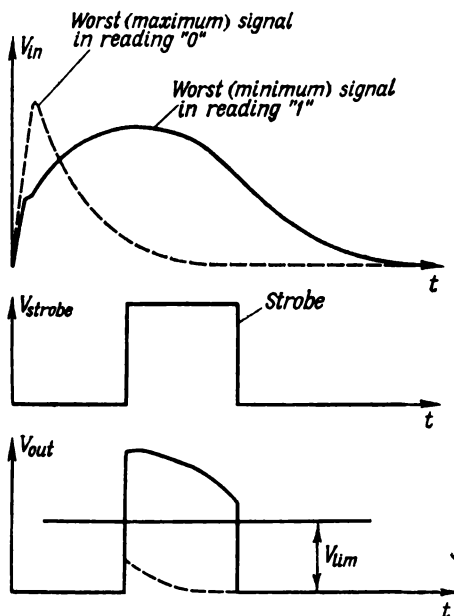


Fig. 7.14

gates, the timing generator produces a read pulse which goes to the respective drivers, DR1 and DR2. These generate signals of

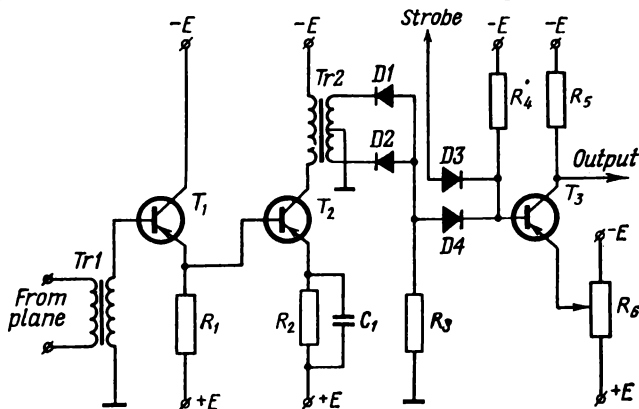


Fig. 7.15

requisite waveform and power to switch the state of the selected cores. This switching action induces in the sense line signals which are routed to the appropriate read amplifiers, and these

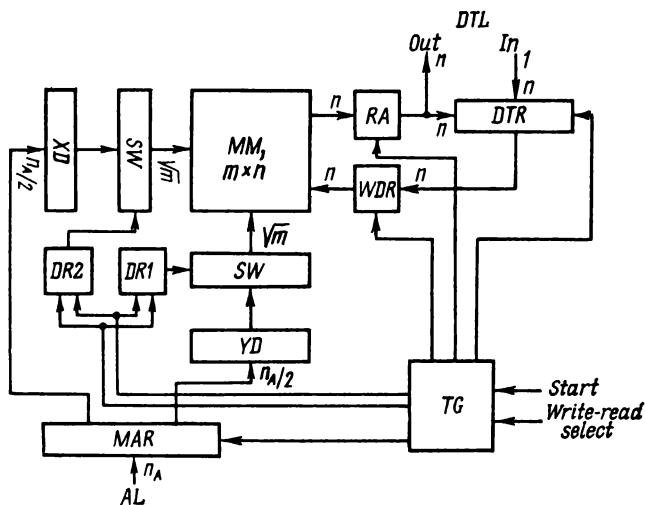


Fig. 7.16

separate valid signals from noise, give them proper shape and put out of the memory unit. The output signal is strobed at the instant when the signal-to-noise ratio is a maximum. This is done by a strobe pulse supplied by the timing generator.



When information is to be written into the memory unit, the first step is a read operation whose function is to clear the selected memory location. During this operation, no strobe pulse is generated and no output signals appear at the amplifier output. A signal from the timing generator causes the drivers to generate write signals needed to switch the state of the selected cores. The data items to be stored are routed by the data transfer line, *DTL*, to the data transfer register, *DTR*. Upon arrival of an appropriate signal from the timing generator, the write drivers, *WDR*, generate bit write control signals according to the binary state of the flip-flops in the data transfer register. If readout is destructive, the information is written back, or regenerated, in about the same manner as during a normal write operation, except that the data item to be written back is taken from the read amplifiers rather than from the data transfer line. A write or read operation is initiated by a START instruction supplied by the central control unit, following the number (word) to be stored, the address, and the write/read select signal.

*Control of fixed (read-only) memory.* A fixed (read-only) memory may be looked upon as a device converting an address into a data item (number) in a manner fixed for each location during the design of the machine. Although the manner in which data are initially read in is important as regards the ease and speed of changing the information stored (mechanical methods would be more time and labour consuming than electrical), no data are written into a fixed memory, except when the computer program has to be changed.

As an example of a read-only memory, we shall discuss a transformer-type unit with a capacity of 1K (1024), 12-bit words, shown in block-diagram form in Fig. 7.17. The number (address) of the memory location from which a word is to be retrieved is transferred to the memory address register, *MAR*, which has ten bits. All bits of the address are transferred into the memory address register in parallel over address lines, *AL*, of which there are as many as there are bits in the address register. The memory locations are divided into four equal groups, each containing 256 locations. A "start" command supplied externally causes three decoders, *X*, *Y* and *Z*, to decode the address. The *X* and *Y* decoders (*XD* and *YD*, respectively) select one of the 256 output lines of the switch (*SW*). The read driver, *RDR*, generates a pulse of appropriate power and waveform which is steered by the switch into the selected wire. The word to be retrieved is read out of one memory location in each group. However, the *Z*-decoder (*ZD*) whose outputs control four groups of coincidence circuits (12AND) allows data to be transferred out of only one group of memory locations, that is, from the selected memory location. The

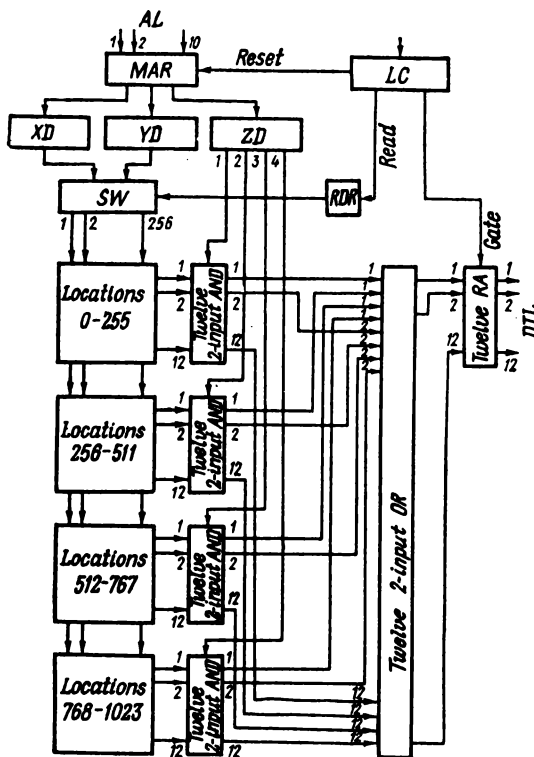


Fig. 7.17

retrieved data are then routed in parallel form through OR gates (12OR) to read amplifiers (12RA) which boost the signals in power and amplitude, filter out noise, and shape the output as appropriate. From the read-only memory, the data transfer lines take the retrieved data to other units within the computer. The necessary control signals which direct operation of the read-only memory are supplied by a local control, LC.

**Control of external memory.** The storage medium in an external memory may be magnetic tape, magnetic drums or magnetic discs. Transfer of information between external memory and internal memory is in arrays each containing a certain number of  $n$ -bit binary data items. On tape, a drum or a disc, each array occupies what is known as an *area*. Each area is assigned a number. The complete range of area numbers defines the total number of addresses that external memory has. During a write or a read cycle, the external memory is given the address of the area where data

are to be written or retrieved. In the general case, the desired area will not be positioned at the heads, and some time will be needed to search for and position it. The time interval involved, known as the seek time, is random, but it cannot exceed a maximum value decided by the number of areas and magnetic drum design. The manner in which data are written on the magnetic surface, the marks applied to label the start of a revolution, the start of an area and the angular positions of each data item within each area are shown in diagrammatic form in Fig. 7.12.

We shall examine control of an external memory unit by an electronic circuit, such as shown in Fig. 7.18, taking a magnetic-drum external memory as an example. The unit is instructed either to write or read data by signals appearing on the WRITE or READ input control wires. In response to a WRITE signal, the memory unit accepts externally supplied data in parallel form over the data transfer line, *DTL-in*, and places them in the area identified by the address supplied externally over the address line, *AL*. The number of words that an area can hold depends on the drum construction and the control circuit. In any case, as many words (data items) will be written during a single write cycle as an area can hold. A READ signal initiates readout of the record stored in the area identified by the address transferred over the address line, *AL*. Now the bits are retrieved in parallel and the words serially, and passed on to the output data transfer line, *DTL-out*.

Apart from the storage medium, an external memory unit includes electronic circuits to accept and handle addresses and numbers (data items), amplify the output signal, select bands and areas, and provide local control. The correct sequence of steps is maintained by the WRITE and READ pulses supplied by the central control and also by signals generated internally in the storage unit, in our case, on three auxiliary tracks.

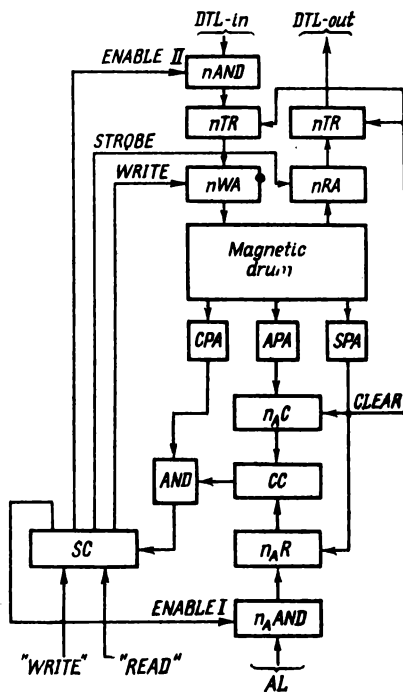


Fig. 7.18

One of the auxiliary tracks carries a start mark producing a signal which, after it is boosted by a start-pulse amplifier, *StA*, is used to clear the registers of the storage unit and the area counter,  $n_A C$ . The second auxiliary track carries the area mark producing a signal boosted by the area pulse amplifier, *APA*. The area counter containing  $n_A$  bits (the number of areas is  $2^{n_A}$ ) counts the pulses arriving at its input, thereby identifying the area within the range of action of the write and read heads. The third auxiliary track carries a location mark which labels each location within an area. The location signals are boosted by a timing pulse amplifier, *TPA*, and utilized to synchronize operation of the storage unit as an array is being written or read.

The writing of an array of data onto the drum is a two-beat process. During the first beat, all registers are cleared by the CLEAR signal supplied by the start pulse amplifier, *SPA*. The area counter  $n_A C$  stores the number of the area passing past the heads, because the counter accumulates pulses coming from the area pulse amplifier, *APA*. In response to the WRITE signal, the local storage control, *SC*, generates an ENABLE I signal which causes the address of the selected area to be transferred from the address line, *AL*, into the address register,  $n_A AR$ , via  $n_A$  two-input AND gates ( $n_A AND$ ). The address of the desired area taken from the address register,  $n_A AR$ , and the address of the current area registered by the angular-position (area) counter,  $n_A C$ , are compared by a coincidence detector, *CD*, which feeds an enable signal to the AND gate only if the two addresses are the same. Thus, the duration of the first beat depends on the setting of the angular-position (area) counter,  $n_A C$ , at the instant when the WRITE signal is applied. During the second beat, the write operation proper takes place. Signals from the clock pulse amplifier, *CPA*, are routed via the AND gate, to the storage control, *SC*, which distributes in synchronism an ENABLE II signal to place data into the input data transfer register,  $nTR$ , via  $nAND$  gates ( $nAND$ ), a WRITE signal to the write amplifier,  $nWA$ , to store the data contained in the input data transfer register,  $nTR$ , and a CLEAR signal to set the input data transfer register,  $nTR$ , to zero. The storage control ceases supplying the ENABLE II, WRITE and CLEAR signals as soon as the selected area has been filled in. When this occurs, an INHIBIT signal goes from the coincidence detector to the AND gate.

A read operation is initiated by a READ instruction. Its first beat is the same as that of a write operation. During the second beat, however, the storage control, *SC*, only generates a STROBE pulse to separate the valid signal from noise. This signal causes the read amplifiers,  $nRA$ , to switch the flip-flops in the output

data transfer registers,  $nTR$ . The binary states of the flip-flops are applied as voltage levels over the output data transfer lines, *DTL-out*, to the internal (primary) storage of the computer.

### 7.3. Input/Output Equipment

**7.3.1. General.** The input/output equipment of a digital computer is the primary link between the operator and the machine. Where data are fed into a computer manually, use is made of key-driven devices. In automatic schemes, this is done with the aid of various input media (punched cards, punched tape, etc.). With a special-purpose digital computer, input and output information is usually presented in analog form. This necessitates the conversion of analog to digital and digital to analog data by means of devices known as analog-to-digital,  $A/D$ , and digital-to-analog,  $D/A$ , converters. In such cases, converters supply a link between a digital computer and the associated analog devices.

Recent years have seen a growing use of  $A/D$  converters in data-logging systems, especially of digital-display instruments which accept analog data display or print output data in digital form.

**7.3.2.  $A/D$  and  $D/A$  converters.** Conversion of analog data into digital form involves what is known as *digitization*. By this technique, a continuous quantity of interest,  $A$ , is represented by discrete values out of a set defined in a particular range. The continuity of amplitude may be removed by amplitude quantization, or simply quantizing, and the continuity with time by sampling.

In quantizing, the precise values of the input signal,  $A$ , are replaced with certain approximate discrete values,  $A_i$ , such that the difference between two adjacent discrete values is an elementary amount, or quantum,  $\Delta A$ . The jumps between values occur at arbitrary times.

In sampling, a sequence of sample values are derived from the continuous signal by observing its amplitude at fixed instants,  $t_i$ . These samples may be, say, the ordinates of points  $a, b, c, d, e$  and  $f$  of the curve  $A = f(t)$  shown in Fig. 7.19. As a rule, digitizers both sample and quantize analog quantities, producing a sequence of sample values each of which has been selected from a discrete set as shown in Fig. 7.19. Digitization may be carried out at a constant quantizing step,  $\Delta A$ , and a constant sampling

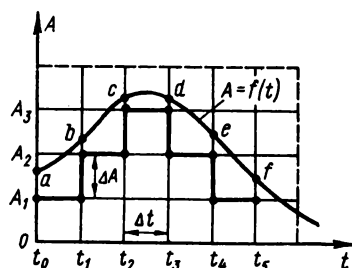


Fig. 7.19

period,  $\Delta t$ , or it may be performed with both factors varied. Most often, use is made of the former technique.

### Digital-to-Analog (D/A) Converters

One form of D/A converters is a digital-to-voltage converter which is a linear gated resistance network, GRN. Its fixed resistors  $r_i$  ( $i = 0, 1, 2, \dots, n$ ) are connected in series (Fig. 7.20a)

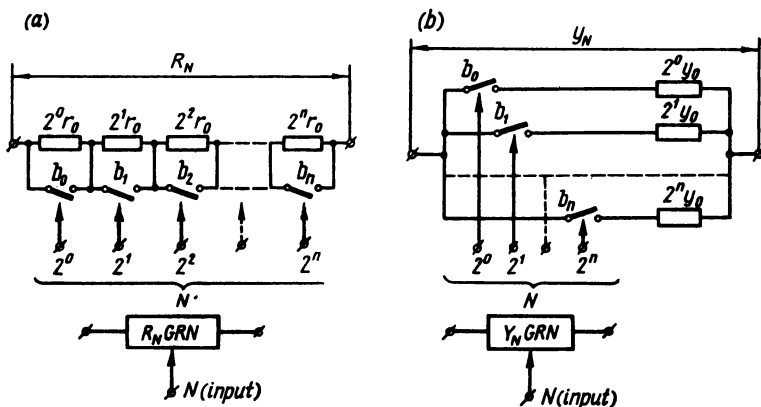


Fig. 7.20

or in parallel (Fig. 7.20b) and are shorted out by switches as may be dictated by the digital data being converted. The resistance of the reference resistor for a binary series-connected gated resistance network is defined as

$$r_i = r_0 2^i \quad (7.1)$$

where  $r_0$  = constant is the fixed scaling resistor associated with the least significant bit.

The total resistance of such a gated resistance network is given by

$$R_N = b_0 r_0 + b_1 r_1 + b_2 r_2 + \dots + b_n r_n = r_0 \sum_{i=0}^n b_i 2^i = r_0 N \quad (7.2)$$

where  $b_i = 1$  when the switch in a given bit position is open (the bit stores a 1),  $b_i = 0$  when the switch in a given bit position is closed (the bit stores a 0), and  $N$  is the input binary number.

For a parallel-connected resistance network, we have

$$y_i = y_0 2^i \quad (7.3)$$

$$y_N = b_0 y_0 + b_1 y_1 + \dots + b_n y_n = y_0 \sum_{i=0}^n b_i 2^i = y_0 N \quad (7.4)$$

where  $b_i = 1$  when the switch in a given bit position is closed (the bit stores a 1),  $b_i = 0$  when the switch in a given bit is open (the bit stores a 0),  $y_i$  and  $y_N$  are the bit and total conductances. The switches controlled by a register or some other digital device may be electromechanical relays, valve or transistor switching circuits.

In parallel-connected gated resistance networks, one-position switches may be replaced with two-position devices. Then the network will have three lines:  $A$  (the "+" side of the source),  $B$  (the ground), and  $C$  (the common return). When the input of the network accepts a binary number  $N$ , the resistors  $r_i$  in the bit positions storing 1's will be connected to line  $A$ , and the resistors in the bit positions storing 0's will be connected to line  $B$ .

It is possible to construct a gated resistance network for any binary number,  $N$ . If the input binary number varies as a function of some specified variable, a nonlinear gated resistance network will result, whose behaviour is described by the characteristic  $R_N = r_0 f(N)$ , where  $f(N)$  is the function being realized for discrete values of the independent variable.

In the simplest, most accurate and reliable manner, digital data are converted to voltage by  $D/A$  converters built around opamps. Operation of such a  $D/A$  converter is based on the fact that the gain of a scaling opamp is incremented in proportion to the input binary number as the resistance of the input resistors is changed. The circuit of a  $D/A$  converter operating on this principle is shown in Fig. 7.21a. Arrival of pulses from the "1" outputs of flip-flops  $FF_i$  causes the electronic switches  $SW_i$  to complete the input circuits of the gated resistance network,  $GRN$ . This arrangement is known as a current-summation network. The summing opamp accepts calibrated voltages because the resistors  $r_i$  are selected to satisfy Eq. (7.1). The output voltage of the converter is

$$V_{out} = - \sum_{i=0}^n b_{n-i} \frac{R_0 V_0}{r_0 2^i}$$

at  $r_0 = R_0$

$$V_{out} = - \sum_{i=0}^n b_{n-i} \frac{V_0}{2^i} \quad (7.5)$$

where  $b_{n-i}$  is a digit (0 or 1) in the input binary number. Va-

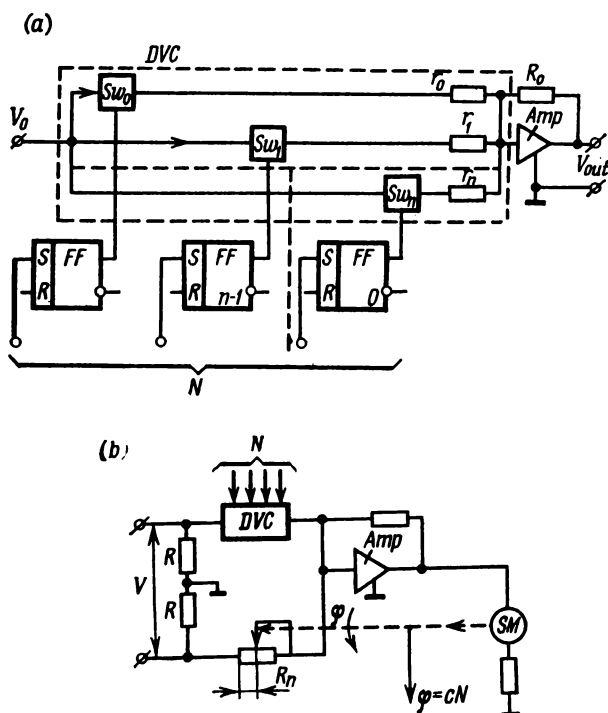


Fig. 7.21

riable resistors,  $r_i$ , may also be placed in the feedback path of the opamp.

Existing digital-to-voltage converters have an accuracy of up to 0.01% and a conversion time of 0.1 to 1  $\mu$ s. Decimal numbers can be converted to voltage by circuits based on a digital potentiometer. An appropriate circuit mechanizing the relation given by Eq. (2.101) is shown in Fig. 2.36. For  $V_{in} = V_0 = \text{constant}$ , we have

$$V_{out} = \alpha V_0$$

where  $\alpha = 0, \alpha_1\alpha_2\alpha_3$  is the decimal number whose magnitude is changed by varying the dial setting in each decade of the potentiometer.

A binary number can be converted to a shaft position, using an arrangement such as shown in Fig. 7.21b, where a bridge circuit is connected to the input of an opamp, and one of the bridge arms contains a digital-to-voltage converter, DVC, such as a parallel-connected gated resistance network. When the bridge is



out of balance, the resistance,  $R_p$ , of the potentiometer does not represent the input  $N$ , and this causes the servo motor,  $SM$ , to move the potentiometer wiper until balance is restored. When this happens,

$$R_p = R_N$$

Since  $R_p = c_1\varphi$  and  $R_N = r_0N$ , the position of the shaft in the servo motor will be defined as

$$\varphi = \frac{r_0}{c_1} N = cN \quad (7.6)$$

where  $r_0$  is the scaling resistor of the digital-to-voltage resistance network,  $c_1$  and  $c$  are proportionality factors.

### *Analog-to-Digital (A/D) Converters*

There are three general classes of methods for converting analog data into a digital form:

(1) The continuous comparison method (used solely in voltage-to-digital conversion) consisting in that the input voltage to be converted is compared with discrete reference voltage settings, and the sum of the latter is incremented until it equals the input voltage. It may be realized with feedback and non-feedback circuits.

(2) The incremental method by which the converter generates a signal each time the input is changed by an incremental amount, and digital representation of the analog input is obtained by summing these incremental signals.

(3) The total value method by which an analog-to-digital converter gives a complete new value, with digits obtained in each bit position. It is mainly used for converting shaft or linear position to digital representation.

The circuit of a voltage-to-digital converter based on the successive approximation variety of the comparison method is shown in Fig. 7.22a. The digits of the output number are generated sequentially, starting with the most significant bit. The input voltage,  $V$ , to be converted, is compared with reference voltages

$$V_i = k2^{n-i}$$

where  $k$  is a constant and  $i$  may take any integer value, 0, 1, 2, ...,  $n$ . At  $k = 1$  and  $V_{\max} \leq |\pm 100 \text{ V}|$ , the reference voltages are 64 V, 32 V, 16 V, 8 V, 4 V, 2 V, and 1 V. The summing opamp is fed a voltage  $V$  (via resistor  $r_0$  equal to the scaling resistor  $r_0$  of a gated resistance network) and a negative voltage  $V_0$  via the resistors in the gated resistance network, whose sum answers Eq. (7.1).

A static pulse distributor, *SPD*, driven by a pulse generator, *PG*, gates out pulses sequentially to outputs 1 and 1', 2 and 2', ...,  $n+1$  and  $(n+1)'$ . When the first pulse, 1, is gated, the flip-flop  $n$  sets and opens the gate in the input circuit of the

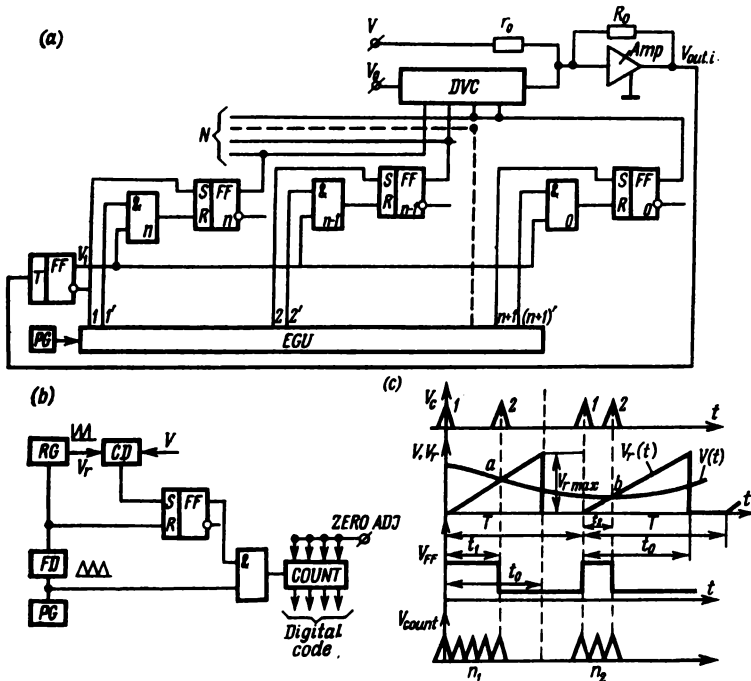


Fig. 7.22

gated resistance network containing the resistor  $r_0$ . The adder delivers an output voltage

$$V_{out\,i} = -\frac{R_0}{r_0} (V - V_0) = -(V - V_0)$$

at  $R_0 = r_0$ . This voltage is fed to a flip-flop which delivers an output signal,  $V_1 > 0$  at  $V_{out\,i} > 0$  ( $V < V_0$ ), and  $V_1 < 0$  at  $V_{out\,i} < 0$  ( $V > V_0$ ). The AND gates open at  $V_1 > 0$ . If  $V > V_0$ , that is, if  $V_1 < 0$ , then arrival of the next pulse at output 1' will cause the AND gate  $n$  to close, and the flip-flop  $n$  will remain in the "1" state. Similar events take place when pulses arrive at the remaining outputs of the pulse distributor. At  $R_0 = r_0$ , we have

$$V_{out\,i} = -V + \sum_{i=0}^n b_{n-i} \frac{V_0}{2^i} \quad (7.7)$$

At the end of a comparison cycle, all bits of the converter will hold  $V_{out\ i} = 0$ , because the voltage  $V$  will be completely balanced out by the voltage  $-V_0$ . Then, by Eq. (7.7) we shall have

$$V = \sum_{i=0}^n b_{n-i} \frac{V_0}{2^i}$$

where  $b_i = 1$  if the respective electronic gate is open and  $b_i = 0$  if closed. After it has received  $2(n+1)$  pulses, the flip-flop register will hold the binary number representing the voltage  $V$ .

The accuracy of this circuit is 0.1% and its rate is up to 200,000 conversions per second at  $n+1 = 10$ .

The incremental (ramp) method is most often implemented by *pulse-counting circuits* such as shown in Fig. 7.22b. In fact, the voltage is first converted to a time interval and then to digital form by counting the pulses filling the interval. The voltage to be converted,  $V$ , and a reference voltage,  $V_r$ , supplied by a ramp generator,  $RG$ , are applied to a coincidence detector,  $CD$ . The ramp generator is triggered by a frequency divider,  $FD$ , which at the same time sets the flip-flop  $FF$  to the 1 state. During the time interval  $t_1$  (Fig. 7.22c) from the instant the ramp generator is triggered and until coincidence is detected between the input voltage and the ramp function,  $V = V_r$ , the gate remains open, and the counter accepts pulses at a controlled frequency,  $f_0$ , from a pulse generator,  $PG$ . The number of pulses registered by the counter is proportional to time. At  $t = t_1$ , the coincidence detector generates a pulse which resets the flip-flop to the 0 state to stop the counting cycle, and no more pulses will be fed to the counter until the time  $t = T$  when the events are repeated all over again. As a result, at times  $t = t_1, t_2, \dots$ , corresponding to points  $a, b, \dots$ , the counter displays numbers

$$n_1 = t_1/T_0 = t_1 f_0$$

$$n_2 = t_2 f_0, \dots$$

The ramp voltage is defined as

$$t_1/t_0 = V/V_{r\ max}$$

where  $t_0$  is the duration of the ramp. Then

$$n_1 = t_0 f_0 V/V_{r\ max} = c_1 V \quad (7.8)$$

that is, the binary number left in the counter represents the magnitude of the input voltage at the time the counting cycle is completed.

Of late, an increasing use has been made of the *frequency conversion method*. A controlled pulse generator,  $PG$  (Fig. 7.23a),

generates pulses at a frequency  $f_p$  proportional to the input voltage,  $V$ , to be converted. The electronic gate,  $EG$ , is caused to open and close by a control circuit,  $CC$ , which is in turn timed by a clock,  $Cl$ . When left open for the time interval set by the clock, the electronic gate lets pulses through so that their number is proportional to  $V$ . At the end of the time interval, the counter is cleared and the counting cycle is repeated.

The utmost in accuracy is obtained with continuous balance feedback  $A/D$  voltage converters (Fig. 7.23*b*). In this method, the magnitude of the input voltage,  $V$ , is compared with a feedback voltage,  $V_{fb}$ . At  $V \neq V_{fb}$ , the difference signal,  $\Delta V$ , causes the AND gate to open and pass pulses from a pulse generator,  $PG$ ,

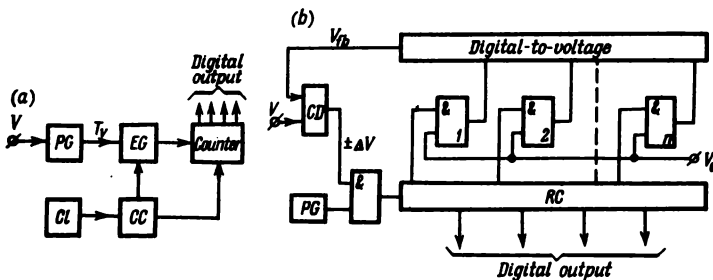


Fig. 7.23

on to a reversible counter,  $RC$ . The AND gates,  $AND_1$  through  $AND_n$ , whose state is controlled by that of the bits in the reversible counter,  $RC$ , let  $V_0$  pass to a digital-to-voltage converter, thereby causing  $V_{fb}$  change in steps until coincidence between the input and feedback voltages,  $V = V_{fb}$ , is detected. At that instant, the AND gate will be caused to close, the binary number in the reversible counter will be proportional to  $V$ .

*Shaft-position converters* (also known as shaft digitizers) usually operate by the total-value method. Shaft position may be converted to digital form by a code wheel (coding disc). A code wheel, or coding disc (Fig. 7.24*a*) has annular zones, each zone representing one bit of a binary number. Within each zone there are conducting and nonconducting segments, representing binary 1's and binary 0's, respectively. These segments are sensed by brushes, and the total signal thus read out gives the binary number corresponding to the position of the shaft coupled to the code wheel. Each time the shaft turns through an angle of  $2\pi/16$ , a new segment on the code wheel comes under the brushes, and a new binary number is read out.

As an alternative, the code wheel may have transparent and opaque or magnetized and unmagnetized segments instead of con-

ducting and nonconducting, to represent the binary 1's and 0's. Then reading will be accomplished by photo-cells, magnetic heads, etc.

A problem associated with code wheels using straight binary codes is the ambiguous reading which may result when two or more bits need to change at the same time, as the brushes move from one segment to the next. Even a slight offset in the brush positions or in the segments would give an incorrect reading. There are several methods to prevent this ambiguity. One is to use the Gray or cycle permuted (or simply cyclic) binary code. With the Gray code, the digit on the right of a 1 is replaced with its complement. For example, the number 0110 in the straight

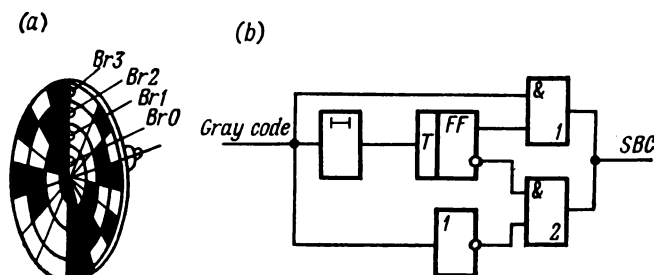


Fig. 7.24

binary code will be 0101 in the Gray code; the number 1011 in the straight binary code will be 1110 in the Gray code, etc. The general rule for this code conversion is: if the number in the straight binary code is  $a_n a_{n-1} \dots a_2 a_1 a_0$  and the same number in the Gray code is  $b_n b_{n-1} \dots b_1 b_0$ , then  $b_n = a_n$ ,  $b_i = a_i$  if  $a_{i+1} = 0$  and  $b_i = 1 - a_i$  if  $a_{i+1} = 1$ , where  $i = 0, 1, 2, \dots, n-1$ .

Since with the Gray code, a transition from one binary number to the next causes only the least significant bit to increment by one, the conversion error does not exceed unity in the least significant bit.

A number in the Gray code can be converted to that in the straight binary code by the rule:

$$a_k = \sum_{i=k}^n b_i$$

that is,  $a_n = b_n$ ,  $a_{n-1} = b_n + b_{n-1}$ ,  $\dots$ ,  $a_0 = b_n + b_{n-1} + \dots + b_1 + b_0$ . Or, in words, the digit in the most significant bit remains unchanged, while the digit in any next less significant bit,  $k$ , remains unchanged on the condition that all the preceding more significant bits contain an even number of 1's; however, it

is replaced with its complement if the more significant bits contain an odd number of 1's. For example, the binary number 10011 in the Gray code will be 11101 in the straight binary code.

Numbers in the Gray code can be converted to those in the straight binary code by the circuit shown in Fig. 7.24*b*. The number in the Gray code is read, with its most significant bits first, through a delay network, *DN*, into a *T* flip-flop and, simultaneously, to an AND gate, AND-1, directly, and to another AND gate, AND-2, via an inverter. Initially, that is, in the 0 state of the *T* flip-flop, AND-1 is open and AND-2 is closed. The *T* flip-flop changes state each time a 1 is applied to its input. On receiving an odd number of 1's, the flip-flop will be in the 1 state, and the digit in the next bit of the number in the Gray code will be passed on to the output via the inverter and AND-2, being converted to its complement in the inverter. On receiving an even number of 1's, the flip-flop will be in the 0 state (AND-1 open and AND-2 closed), and the digit in the next bit of the number in the Gray code will pass on to the output via AND-1, with no inversion.

### 7.3.3. Input Media and Devices

#### *Input Media*

Most often, input media for digital computers are punched cards, punched paper tape, and magnetic tape. The punched card shown in Fig. 7.25*a* is made of dense elastic paper. It has 80 columns and 12 horizontal punching positions. Source data are punched into a card as rectangular holes at the intersections of columns and positions. Information can be punched in a variety of codes. For example, a decimal number may be arranged on a single horizontal line, using the binary-coded decimal (*BCD*) representation. Then, a binary 1 may be represented by a hole and a binary 0, by no hole. Each digit of a number occupies four columns and is coded in the 8421 code. The spacing between adjacent digits is one column wide. The columns at the beginning of a punched card are usually reserved for auxiliary information, such as the task number, the card number, markers, etc.

Standard punched (or perforated) tape (Fig. 7.25*b*) is fabricated from paper 0.1 mm thick impregnated with mineral oil and usually has five tracks, channels or levels, for information bits and one feed or sprocket track so that the tape may conveniently be pulled through. Each digit of a decimal number is represented in the *BCD* form as holes in the "8", "4", "2" and "1" levels (in the figure, this is shown for the digits 1 through 6). If a digit of a number has been punched into a given level or track, an additional ("identifier") hole is punched in the lowermost level; if

an instruction has been punched, no "identifier" hole is provided. There are also six-, seven- and eight-track tapes. Six- and seven-track tapes are 22.5 mm wide, and eight-level track, 25 mm wide.

More seldom, use is made of punched tape prepared from cine film, with information bits punched at the various levels as rectangular holes.

In contrast to punched cards, punched tapes do not allow the information they store to be sorted or collated, unless the tape is cut, which is a major disadvantage of punched tape.

Magnetic tape used in computers generally consists of a plastic base of about 40 to 60  $\mu\text{m}$  thick and a magnetic coating applied to one side (or, although more seldom, to both sides). Data are recorded on magnetic tape by producing small magnetized areas.

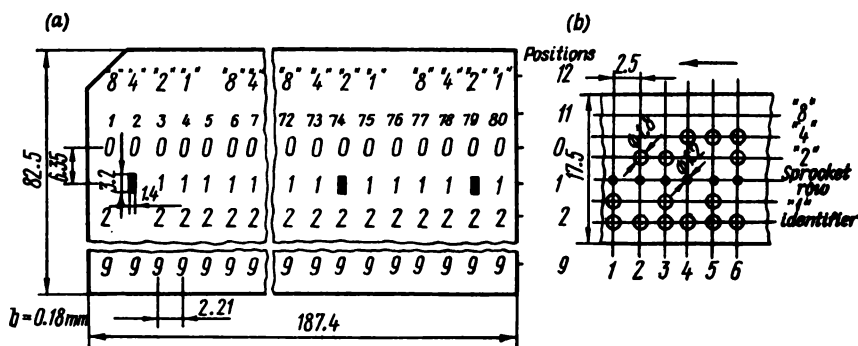


Fig. 7.25

Owing to high recording density, magnetic tape can store large blocks of data within a limited space and permits much higher data transfer rates than other storage media at the same mechanical speed. Also, magnetic tape can repeatedly be used for recording new data as the old information is erased.

### Transfer of Data to Storage Media

The holes representing information are made in cards and paper tape on devices called card punches or tape punches. The manner in which this is done by a parallel punch is clear from Fig. 7.26a.

Source data are keyed in, automatically encoded and transferred as electrical signals by a keyboard unit, *KU*, to a punching station. The keyboard unit is made up of a keyboard and a diode decoder, *DD*. The keyboard has keys to handle ten-decimal digits (0 through 9), of which there are as many as columns on a punched card or code channels or levels on a punched tape, and also keys for characters and control. Source data are usually keyed in on the keyboard in decimal notation, while instructions and ad-

dresses in octal notation, starting with the most significant bit. Pressure on a key closes a contact set, and this completes appropriate circuits in the diode encoder, while the key stays "put" until a hole is punched. After all the bits are keyed in, the release key is pressed, and the diode encoder converts the decimal or octal codes into a *BCD* one (for the mantissas or fractional parts of numbers) or binary form (for the exponents, instructions and addresses). From the outputs of the diode encoder, the signals are routed to the electromagnets, *EM*, of the punching station. In operating, the electromagnets drive the punches, and these punch holes, all at a time, across the width of a card or tape. After a number has thus been punched, the card or tape is advanced to the next punching position.

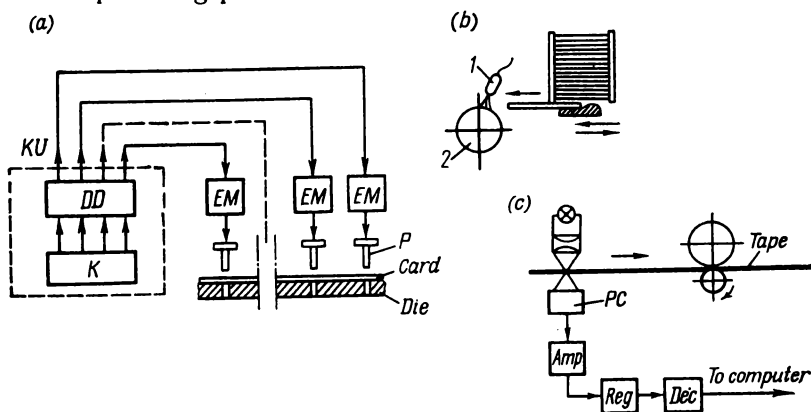


Fig. 7.26

Often, data are punched by the area. In such a case, the keyboard has a limited number of figures sections (nine and more). If use is made of 80-column cards and data are in decimal notation, serial keypunches are employed. Each keypunch has 12 punches (as many as there are positions in each column), and holes are punched in a column as a key is pressed, after which the card is moved, and the next column is positioned under the punches.

Information can be punched onto paper tape on a standard teletypewriter, such as the Soviet-made CTA-2M, with a perforator attachment, or by special-purpose tape punches.

As already explained, the tape is punched in all positions across its width at a time, as trains of pulses are fed from the encoder to the electromagnets of the punches installed opposite each of the five, six, seven or eight code levels and one feed or sprocket hole. Punching is carried out in a start-stop manner, with the



tape held stationary while holes are punched. The punching rate is about 20 characters per second (for tape punches such as the Soviet-made ПЛ120-2 and У-535).

The accuracy of the data punched into a card or tape is checked by devices called verifiers. In principle, a verifier is not unlike a punch, except that it has a mechanism to sense punched holes instead of a punching mechanism. After a card or tape has originally been punched, it is sent to a verifier. Here, a second operator keys in the same information as the original operator, and this is compared with the recorded data. If there is a disagreement, the verifier issues an error warning. Often, the second operator punches a second set of cards or tapes, and both sets are then checked by a verifier.

### *Input Devices*

Source data may be entered in a digital computer either by means of storage media or directly from a controlled plant or system, source documents, or by voice. In practice, the most commonly used method of data entry is from storage media, and a good deal of research has been conducted in this field. Data entry from source documents and by voice, although extremely promising, has not yet been developed sufficiently for general use.

Among the requirements that an input device should meet are speed and reliability. The information keyed into cards or paper tape can be read either by the electromechanical (electric contact) method or by the photo-electric method. Punched cards are usually read by the electric contact method. In this method, a card is passed (Fig. 7.26*b*) under a set of metal brushes, 1, which "sense" all the 80 columns of the card. When a brush reaches a punched hole, it drops into the hole and touches a contact roller, 2. This produces a pulse which after amplification and decoding, is applied to the input of the computer. The input device is linked to the computer by multi-way connectors in all the eighty channels. Cards are read at a rate of 700 or more per minute.

In the photo-electric method which uses photo-diodes for hole sensing, punched cards can be read either a column at a time (that is, all the positions in a column are read simultaneously) or by the row or position (in which case, the same position in all the 80 columns is read at a time). In the former case, cards are read at a relatively low rate (about 300 cards per minute). In the latter case, the reading rate is higher, but the device is of a more complicated design.

Punched paper tape can be read by the electromechanical (electric-contact) sensing unit of a teleprinter. Each row (character) on the tape is sensed by five pins or probes located each opposite

one of the levels or channels. When the probes find holes in the paper tape, they protrude through the holes, and a combination of electric pulses is generated, representing the digit punched into that row. The tape is advanced by a tape transport which stops each time a character is read. The reading speed is up to 60 characters (rows) per second (for example, the Soviet-made CY-1 tape reader). Preference is however given to photoelectric tape readers which have reading speeds up to 1500 characters per second, with an added advantage that the storage medium need not be made strong mechanically. Where data have to be read from five-, six-, seven- or eight-level tape alternately, the reader can be adapted to a particular tape type by changing the reel hubs and by adjusting the tape-width limiter. The key component of a photoelectric reader is a photoreading head which consists of light sources (Fig. 7.26c shows a light source for one code channel) and photo-diodes, *PD*, each opposite one code channel and the feed channel to read sync (timing) pulses. The tape is advanced between the light sources and the photo-diodes. When a hole occurs in the path of the light beam from the respective light source, light illuminates the photo-diode, and a pulse is generated. Boosted by an amplifier, *Amp*, the pulses are applied to the input register, *Rg*, via gates which open at the instants when a number (data item) is read and set the flip-flops to the 1 state. As a result, the register stores the number read from a given row, which is then routed to a decoder, *DC*, and the storage unit of a computer.

Preparation of punched cards and paper tape for use in input devices is a labour-consuming operation. Recent years have seen a good deal of effort put into the development of devices that would read in information directly from original documents, using automatic character recognition techniques for this purpose. Most often, the characters on an original document are compared with a master set of standard or reference characters. Reference sets may be made in the form of photographic masks or as electric models of standard characters (resistance matrices, ferrite cores, etc.). In one arrangement an automatic character reading machine uses a mask carrying both a negative and a positive image of each character. Each character in an original document is scanned by a flying spot (which traces out up to 30 lines per character), and is reproduced on a *CRT* face as a positive and a negative image. Then an optical system steers it among several channels, and the scanned character is projected onto a mask with paired reference characters. The scanned positives are then superimposed on the reference negatives, and the scanned negatives on the reference positives for comparison. As a result, each channel generates a light flux of an intensity proportional to the

amount of mismatch between an actual character and its standard representation. Finally, the light flux is converted to an electric signal, and the machine selects the character for which the amount of mismatch is the least.

**7.3.4. Output devices.** Output equipment delivers the results of computer work outside the computer. It may be divided into two broad classes: direct and indirect. Direct output devices produce data in a form that can be used directly by man (printed data from an electric typewriter or a line printer). Indirect output devices produce data in a "machine" language, that is, in a form which is not directly usable to man (punched cards or magnetic tape). Some output devices may be a combination of both techniques.

Consider the design and operation of printers. They may be classed into serial and parallel. The former (such as a standard electric typewriter) are single-action, and only one character is printed at a time. The latter print an entire line of characters at a time (for which reason they are called line-a-time printers or simply, line printers). Another way to class printers is according to the type of motion they use. Those using mechanical motion are called mechanical printers; all others are nonmechanical printers. With mechanical printers, ink applied to tape is transferred as characters onto a printing medium (usually, paper). This may be done by rotating print wheels, drums or matrices. Nonmechanical printers utilize cathode-ray tubes, character-display devices, magnetic tape, etc.

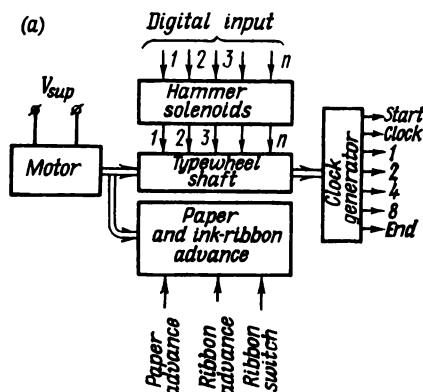
### *Mechanical Printers*

In mechanical printers, printing is done by causing metal characters to leave their impressions on paper over an ink ribbon. The metal characters may move intermittently or continuously. As an example, consider a line printer utilizing continuously rotating print wheels (Fig. 7.27a and b), usually called an *on-the-fly printer* (Fig. 7.27a and b). There is a type wheel, 1, for each bit or character position on the line, rotating at a constant speed. Printing is accomplished by actuating hammers, 2, when the appropriate characters are opposite the paper; the movement of the hammers drives the paper against the type wheels, and the characters are thus produced. The hammers are actuated by solenoids which are energized by properly timed pulses.

Data to be printed are inserted into the printer over  $n$  channels as current pulses synchronized with the position of the shaft carrying the type wheels. Synchronization is provided by a clock pulse generator, CPG. On-the-fly printers can print from 20 lines per second upwards. In addition to numerical data, they can print letters, special symbols and plots (alphameric printers). Soviet-

made on-the-fly printers, such as the ALIY-128-2 or the Y-545, have 128 type wheels (accordingly, they can print up to 128 characters per line) and operate at a speed of 400 lines per minute.

The on-the-fly printing principle is also utilized in a single-wheel printer such as the Soviet-made АПМ-1, which prints



(b)

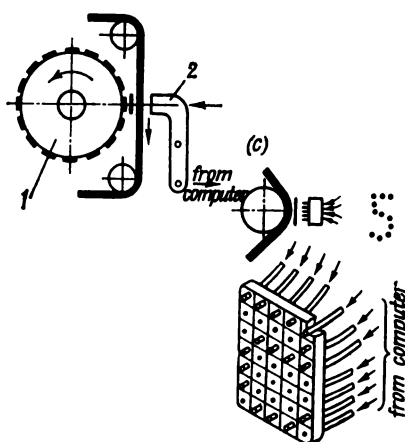


Fig. 7.27

on the paper. As a result, the selected character or numeral (the figure shows the numeral '5') is recorded as a pattern of dots arranged in a matrix. Matrix printers may be of the line-at-a-time type, in which case several matrices will print a line of characters at a time.

characters one at a time, starting with the most significant bit on each line. The carriage moves past the print wheel intermittently, and the solenoid is energized by pulses from the computer, precisely timed with the position of the correct character above the hammer. This type of printer operates at a rate of 10 characters per second.

Single-action printing is done by electric typewriters, such as the Soviet-made ЭУМ-23 or ЭУМ-46. They are usually built in the desk-top models and controlled by electric pulses which energize the solenoids of the appropriate type bars. They can usually print several copies of data in two colours. Printing speed is about 7 characters per second, with up to 162 characters per line, depending on platen width.

**Matrix printers** (Fig. 7.27c) use a matrix of light-weight hammers or wires, each actuated by a solenoid. When the appropriate solenoids are energized by a combination of electric pulses, the associated hammers or wires strike the paper over an ink ribbon and transfer ink

## Nonmechanical Printers

Character (or symbol) display devices produce luminous images of characters by means of cathode-ray tubes, glow-discharge tubes, or pulse tubes. Most promising among them are shaped-beam display tubes of the Charactron type (Fig. 7.28a). In this type of tube, an electron gun generates a diffuse beam of electrons. Symbol-selection deflection plates controlled by a character generator direct the beam towards an appropriate hole in a stencil matrix,  $M$ , which shapes the beam, and the desired character is formed. After leaving the stencil matrix the beam is re-directed by an

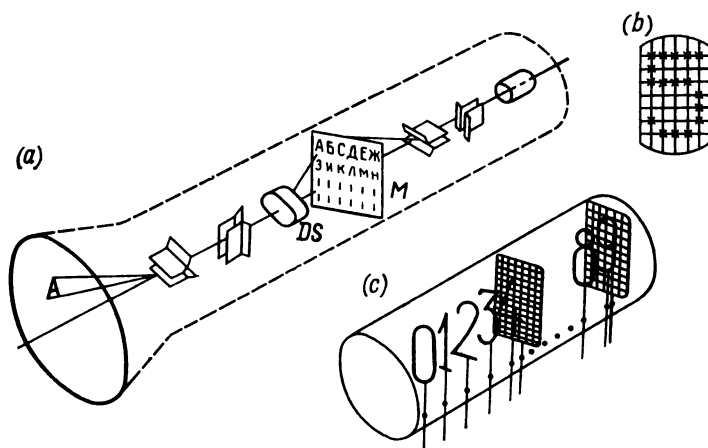


Fig. 7.28

optical system, OS and another set of image-positioning deflection plates towards the desired position on the viewing screen, where the beam is displayed as an image of the desired character. During a writing cycle, the desired characters are displayed on the screen one after another into lines which may be observed visually, photographed, or transferred onto an intermediate medium by an electrostatic process (described below) and then onto paper.

Similar types of a shaped beam tube are the Typotron and the Composatron, with writing speeds up to 20,000 characters per second, or even more.

Relatively simple nonmechanical printers are built with glow-discharge tubes. In one such device, the tubes are arranged in a  $5 \times 7$  matrix, and characters are formed by the glow spots given up by the filaments. In some devices, the matrix can be formed by vertical and horizontal wires (Fig. 7.28b). Each time

a control voltage is applied to a vertical and a horizontal wire, a glow spot appears at their intersection. Combinations of glow spots form characters displayed on a viewing screen with a long after-glow.

Of late, trend has been towards the wider use of numerical readout tubes (Nixies), such as shown in Fig. 7.28c. In a numerical readout tube, the cathodes made of tungsten wire are given the shape of the numerals from 0 to 9, and the anodes are fine-wire meshes. The cathodes and the anodes are enclosed in a neon-filled glass envelope. When a voltage is applied to a cathode and anode, the neon around the cathode gives up a glow which stands out as a visible numeral.

There are several printing methods which depend for their effect on changes in the properties of a storage medium under the action of electric fields. Basically, in a device using such methods, there are two sets of electrodes, one comprising electrodes each having the shape of a character, and the other made up of plane plates. Printing is done on a suitably treated strip of paper passing between the two sets of electrodes. When voltage pulses are applied to electrodes, an electric field the shape of the respective character is set up, and this produces a latent image on the paper.

In electrostatic printers, the latent image of a character is formed as an electrostatic charge directly on a writing medium having high dielectric properties (paper, polystyrene, synthetic materials, etc.), which is then treated to develop and fix the image.

In electrochemical printers, the paper is impregnated with an electrolyte, and an image is formed by a stylus connected to an electric source. The energized stylus causes the electrolyte to decompose, and this leaves traces that are the shape of a character.

In electrothermal printers, latent images of characters formed by an appropriate coating material are made visible by the action of heat, so that the coating darkens or even burns partly to form the desired characters.

Electrographic printers make use of an electric spark for recording images of characters on a medium such as strip paper. The paper may be moved between writing electrodes (in which case, the electric sparks burn holes in the paper in the form of characters) or near writing electrodes (in which case characters are formed of burned spots).

In the magnetographic process, characters may be recorded either directly on a suitable type of paper or on an intermediate medium (say, a magnetic drum) for further transfer to a final medium. The surface of a medium is given a thin coat (5 to 15  $\mu\text{m}$ ), of ferromagnetic powder (such as nickel-cobalt alloy or

iron oxide). Characters are recorded by suitable magnetic heads or styli as appropriately formed magnetized areas. These latent images are then made visible and fixed.

It may be noted that nonmechanical printing methods are gaining more and more ground because they offer high writing speeds, reliability, versatility, silent operation and many other advantages over mechanical printers.

## **7.4. Control Equipment**

**7.4.1. General.** The control unit of an electronic digital computer directs the computer components to perform the appropriate operations necessary to solve a problem automatically, in accordance with a program read into the storage unit.

The component units of any computer operate in response to commands or instructions which are generated in a predetermined sequence and at predetermined instances of time. These instructions are supplied by the control unit. From this view-point, all types of computers are alike as regards their control units. While other computer units (input and output devices, the arithmetic unit, memory) are to a certain degree independent, the control unit is inseparably linked with all the remaining units of the computer. Accordingly, the structure of a control unit depends not only on the class of problems the computer is designed to handle, but also on the performance and capabilities of the other units and the principle(s) underlying the problem-solving procedure(s). This is the reason why computers of the same type may have different control units.

Special-purpose digital computers intended to handle a narrow class of mathematical problems or even to solve repeatedly the same problem for various input data, may have a "rigid" type of control unit, capable of directing the computer on a single-program basis.

Irrespective of whether a computer is general- or special-purpose, it may use central or local control or both. With central control, there is a single central control unit which interprets instructions to provide the signals that cause all computer elements to perform their functions; these command signals go directly to the elements or units concerned. Fully centralized control is advantageous, however, with very simple computers. Most often, central control is supplemented with local control. In such a case, the central control unit only provides signals which time the operation of the computer elements, while the actual operation of a respective unit is directed by its local control. Local control reduces the "burden" on the central control unit and gives the com-

puter elements a degree of freedom which enables them to be sometimes used in parallel.

With either the "rigid" program of a special-purpose computer or the "flexible" program of a general-purpose machine, the control unit goes through a sequence of actions which produce a sequence of command signals. The sequence of periodically repeated actions performed by a computer in response to one instruction or a group of instructions is called the *cycle of the computer*. The duration of a machine cycle is called a *machine cycle time*. Depending on the time allocated to perform a basic operation, there are computers with a fixed and a variable cycle time (or cycle rate). With a fixed cycle rate, the same time interval is allowed for performing any operation; it may be selected to fit either the longest or the shortest operation. In the former case, part of the time will inevitably be wasted in carrying out a shorter operation. In the latter case,  $n$  cycle times will be needed to carry out long operations (where  $n$  is an integer). With a variable machine cycle rate, different time intervals are assigned to different operations, and every next operation commences immediately after the previous one is completed. Computers with a fixed cycle rate are called *synchronous*, and those with a variable cycle rate, *asynchronous*. A synchronous computer may have fully centralized control, while an asynchronous machine will usually have a combination of central and local controls, with local controls taking care of the longer operations, and central control, of the shorter operations. There are also synchronous-asynchronous computers. In them, the execution of some instructions is accompanied by an interruption of the basic machine cycle to carry out some additional actions determined by the type of instruction. After these additional steps are executed, the computer resumes the basic machine cycle.

For a single-address digital computer, the steps of a basic cycle are:

- (a) access to memory to fetch an instruction;
- (b) access to memory to fetch the number to be operated upon (the operand);
- (c) execution of the operation.

The steps of a basic cycle for a two-address digital computer are:

- (a) access to memory to fetch an instruction;
- (b) access to memory to fetch the first operand;
- (c) access to memory to fetch the second operand;
- (d) execution of the operation.

These are direct cycles. There are also so-called overlapping cycles in which the computer executes an instruction fetched from memory during the previous cycle, while at the end of the current



cycle the control unit fetches an instruction to be executed during the next cycle.

**7.4.2. Central control.** An illustrative example of the functional elements that make up a central control unit is offered by that of a three-address parallel digital computer. Its direct basic cycle involves the following steps:

- (a) access to memory to fetch an instruction;
- (b) access to memory to fetch the first operand;
- (c) access to memory to fetch the second operand;

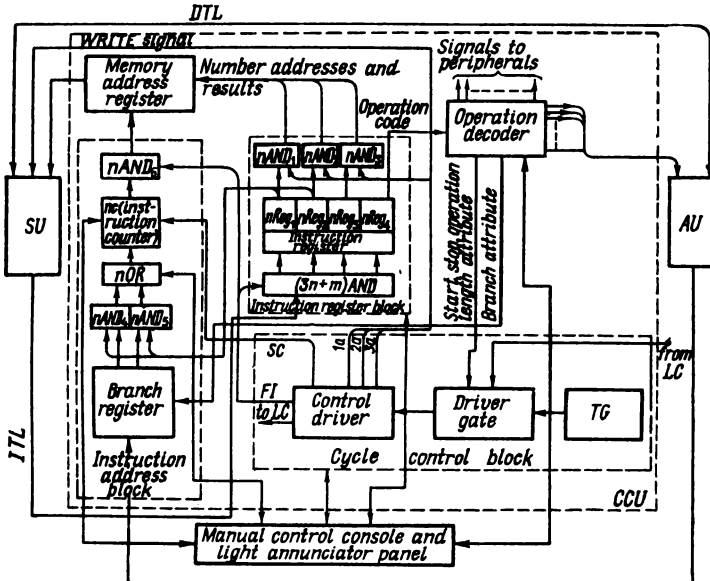


Fig. 7.29

- (d) execution of the operation in the arithmetic unit;
- (e) access to memory to store the result.

To be able to perform the above steps of the basic cycle, the central control unit should include the following functional elements (see the block diagram of Fig. 7.29):

(1) A cycle control unit to generate the basic control signals essential for the computer to go through each cycle step. This unit usually consists of three stages:

a timing (or clock) generator which generates a continuous sequence of pulses and is the principal synchronizing element of the central control unit;

a driver gate to gate pulse trains from the timing generator to a control driver, to interrupt it in the case of some operations, and to gate synchronizing pulses from external devices;

the control driver to generate "fetch instruction" (*FI*) commands, to initiate memory reference at the first address (*A1*), the second address (*A2*) and the third address (*A3*), a signal to start a machine cycle (*SC*), and also command pulses needed by the appropriate local control units.

(2) An instruction address block to generate the consecutive addresses of instructions where these are executed in their natural sequence, and also to hold the first or second address, according to the value of the  $\omega$  signal in the case of a branch instruction (a conditional transfer of control). The key element of this block is an instruction counter.

(3) An instruction register block to hold the instruction currently fetched from memory and to transfer successively the first operand, the second operand and the result to a memory access (or address) register and also to transfer the codes of the operations to be executed to an operation decoder.

(4) An operation decoder to interpret the operation code coming from the instruction register and to gate from one of its outputs a signal to the respective computer device. The operation decoder is an  $m$ -input gate with as many outputs as there are operations performed by the computer.

(5) A memory access register to hold numbers read from memory locations at the addresses specified in instructions and to route them either to the arithmetic unit or to the instruction register, depending on whether a particular address has come from the instruction counter or the instruction register. In most cases, the memory access register is part of the storage (local) control.

(6) The console and light annunciator panel. These are not usually included in the control unit of a computer, although they are essential to its control.

The central control unit shown in the block diagram of Fig. 7.29 operates as follows. The address of the first instruction to be executed is transferred into the instruction counter from the control console or over the branch circuit. An *FI* (fetch instruction) command applied via the AND gates,  $n$ AND-6, causes the instruction address to be transferred to the memory access register, and this takes the desired instruction from the addressed location. Then the *FI* signal causes this instruction to be transferred via AND gates,  $(3n + m)$  AND, to the instruction register. From the register,  $mRg$ , the instruction code is transferred to the operation decoder which sets the arithmetic unit, memory or other computer elements needed to execute the instruction and also generates an operation length indicator signal. This "length" indicator causes the timing generator to feed to the local control an appropriate signal (which may be a voltage level or a number of pulses) corresponding to the time interval needed to execute

the operation or operations involved. The control driver stage then gates out command pulses to re-write the first, second and third address in the instruction. The "A1" command pulse causes the address of the first operand to be placed in the memory access register via the  $n\text{AND-1}$  gate. As a result, the first operand is taken out of memory and routed over the data (or number) bus to the arithmetic register or adder. In a similar manner, the "A2" command pulse causes the second operand to be placed in the arithmetic unit. The arithmetic unit performs the arithmetic operation ordered, after which the "A3" command pulse causes the result to be transferred over the data (number) bus to memory where it is read into the addressed memory location. This completes the execution of the selected instruction of a program. Then the SC (start of cycle) command pulse increments the contents of the instruction counter,  $IC$ , by unity, and after the control driver stage gates out a FETCH INSTRUCTION command pulse, the next instruction of the program is executed in the manner already described. The program execution can be terminated either by a STOP command or a branch instruction which alters the strict sequential stepping in the execution of a program.

Each time the arithmetic unit executes an arithmetic or logic operation, it generates a  $\omega$  signal, which is the attribute of the result. When the result is negative or when a logic condition is not satisfied,  $\omega = 1$ ; in all other cases,  $\omega = 0$ .

The branch instruction selected during a cycle in the instruction register,  $IR$ , is executed according to the value of the  $\omega$  signal produced during the previous cycle. If  $\omega = 0$ , the branch register will enable the  $n\text{AND-4}$  gates, and the address is transferred from the  $nRg-1$  instruction register into the instruction counter,  $IC$ . If  $\omega = 1$ , the branch register enables the  $n\text{AND-5}$  gates, and the address from  $nRg-2$  is transferred into the instruction counter. After the branch instruction has been executed, the computer resumes the sequential stepping from the address transferred into the instruction counter by the branch instruction.

In some cases, the execution of an instruction may be accompanied by transfer of the next instruction into the instruction register (an overlapping cycle).

In two- and single-address digital computers, the central control unit is built along similar lines, but is simpler in design. Above all, it uses a simpler instruction register unit (a two-address machine will have no  $nRg-3$  register and the associated AND gates, while a single-address computer will only have an  $mRg$  and  $nRg-1$  and the associated input and output gates). The simplest control unit of all and the shortest cycle will be found in general-purpose single-address digital computers.

**7.4.3. Effect of the control structure on computer speed.** The speed of a computer may be increased (or, rather, the time needed for problem solving may be cut down) by arranging several steps or operations to be executed simultaneously. As often as not, a control unit will be designed to combine the retrieval of data and instructions, the execution of an arithmetic operation and the instruction fetch phase, the data input-output phase and the actual problem solving, transfer of data between external and internal memory and computational operations.

Incorporation of index registers in the central control unit enables operations on data to be performed concurrently with instruction modification. In such a case, an instruction address is modified in the central control unit, and the arithmetic unit is left free for operations on data (numbers). With index registers in the central control, the instruction format is changed to include, in addition to the usual operation code and addresses, also a bit designating the need for instruction modification and the number of the index register involved. Then the contents of this index register are automatically added to the address part of the instruction, and the instruction thus modified is then executed. At the same time, the contents of the index register are incremented or decremented (usually by unity). Apart from an increase in computer speed, index registers provide a means for keeping track (without a separate counter) of the number of times that the computer has been in the program loop, and also make it unnecessary to restore the program instructions (which is ordinarily provided for by the program) that are changed in the case of program loops, because these instructions will be held in memory in their original form.

The speed of a digital computer may further be enhanced by provision of separate bus systems for referencing the internal memory, several internal memories and arithmetic units which may operate all at the same time independently, with each instruction executed during a single machine cycle.

In computer control applications, it is often necessary to interrupt the solution of one problem (that is, execution of a particular program) and to switch to another problem (program), or put in new data and put out the results, which is often the case with a control computer operating over communication links. Also, the computer or some part of it may require servicing. In all such cases, the need arises for what is known as a program interrupt. This need is satisfied by a program interrupt system. An interrupt phase is usually initiated by an interrupt request, *IR*, which may come from without the computer (the plant or system being controlled) or it may be set up internally (for example, in computers with a built-in check feature, should an error occur in

the problem run). The interrupt request gives the number of a suitable stored program which the computer is requested by the program interrupt unit to execute.

Depending on the form of the interrupt request loaded into the program interrupt unit, the computer will either discontinue the problem run and execute the program specified in the interrupt request, or defer the execution of the interrupt request until the problem being run is carried out to completion. An interrupt request also specifies the form of interruption, that is, whether the results should be saved or destroyed. In the former case, the memory will save all the results obtained prior to the interrupt request, and the contents (codes) of the instruction register, instruction counter, index and other registers will be written into appropriate memory locations so that, after the program called out by the interrupt request is executed, the computer will readily resume the interrupted run. In the latter case, all intermediate results and the contents of all registers and counters of the control unit are lost.

The design of a digital computer is determined not only by the mathematical operations that are to be carried out, but also by the physical facilities needed and by the sequence in which they may or should be arranged to operate. Of necessity, certain sequences of instructions occur repeatedly and cyclically in the solution of a problem, which is also true of sequencing the command steps in the execution of the various instructions. It has therefore been suggested to organize such repetitive sequence as programs in their own right. This technique has come to be known as *microprogramming*, and the steps within such microprograms as *microinstructions*. They are stored in a special memory unit, the microprogram storage or memory.

A computer using microprograms operates as follows. On the basis of the operation code in the instruction set up in the instruction register, the local control reads out of the microprogram memory the first microinstruction. Each microinstruction controls an appropriate functional unit so as to execute the instruction set up on the instruction register. After the first microprogram (the first instruction) has been executed, the next instruction is set up on the instruction register, which calls out of the microprogram memory the respective microprogram, etc. This sequence of events is repeated all over again until all computations are completed or the sequential stepping is interrupted by a jump microinstruction. Jump microinstructions provide a ready means for altering the individual operations while they are being performed.

Microprogramming greatly enhances the flexibility of the computer structure and enables the computer instruction set to be adapted to a particular problem and computer usage.

## Chapter VIII

### Principles of Programming

#### 8.1. Basic Definitions. Hypothetical Computer

**8.1.1. Steps in preparation for problem solving on a digital computer.** Before a workable program can be compiled for solving a problem on a digital computer, one has to have an algorithm for its solution. Thus, the search for an optimum algorithm is the first step in preparation for problem solving. For example, if an engineering problem can be reduced to a set of algebraic or differential equations, its algorithm may well be based on a numerical technique. In selecting which of the numerical methods to use, account must be taken of the accuracy sought, the time needed (or allocated) to solve the problem, and the specific features or capabilities of the available computer.

In most cases, algorithms are specified in terms of an *algorithmic language*, which is a set of symbols and rules adopted for defining the computational process(es) involved.

The second step in preparation for problem solving is coding, that is, the translation of the algorithm written in an algorithmic language into one readable by the computer. This step is in effect the compilation of a program to be executed in terms of the instruction set adopted for a particular computer. This step also involves the allocation of space in the computer memory to the routines, source and auxiliary numbers, and to intermediate results.

The last step in problem preparation is debugging, that is, testing the program, detecting and correcting whatever errors there may be.

**8.1.2. Computer-performed operations and instructions.** The operations performed by a digital computer may generally be classed into:

(a) data transfer between devices and inside the same device, and input-output (I/O) operations;

(b) logical operations such as columnwise logical multiplication and addition, and columnwise modulo 2 addition;

(c) arithmetic operations such as addition, subtraction, multiplication, division;

(d) control operations, including conditional and unconditional transfer of control, starting and stopping the computer;

(e) operations of shifting and normalization.

Present-day digital computers use single-, two- or three-address instructions. Since in the simpler arithmetic operations there are

only two operands and one result, the three-address type of instructions suits best.

**8.1.3. A hypothetical computer.** It is convenient to follow the principles and practices of computer programming by reference to a hypothetical computer. With such a device, the specific features of an actual computer will not mask the general aspects of computer programming.

In the discussion that follows, instructions will have the following format:  $\theta abc$ , where  $\theta$  denotes the operation code (opcode),  $a$  gives the address, or location number, of the first operand,  $b$  that of the second, and  $c$  the address of the result.

Let our hypothetical computer have the following characteristics. It will be a binary machine with a three-address instruction set and floating-point arithmetic. Its internal memory can store 2048 binary numbers, each 43 bits long. Data will be read into the computer from punched cards, and the results will be punched out on cards and paper tape. The allocation of bits to numbers (operands) and instructions within a machine word is shown in Fig. 8.1 and Fig. 8.2, respectively.

For a better understanding of how data are set into the computer, we shall assume that only one operand or one instruction is punched into each card. In fact, many more operands or instructions can be coded on the same card in an actual computer.

When an overflow occurs (or when the result has an exponent exceeding the decimal 19), the machine will stop automatically. If the result has an exponent smaller than  $-19$ , the answer will be a machine zero. In the case of failure to satisfy logical conditions and also when arithmetic operations produce results smaller than zero, these conditions will be flagged by an appropriate attribute,  $\omega = 1$ .

Each operation code includes a check bit. If the check bit of an instruction contains unity, the computer will stop after it has executed that instruction. If the check bit is zero, the computation will continue. This "halt on one" feature is used in debugging and during preventive maintenance.

In discussing the instruction set of our hypothetical computer, the contents of addresses  $a$ ,  $b$  and  $c$  will be designated by the same respective letters enclosed in parentheses:  $(a)$ ,  $(b)$  and  $(c)$ . The addresses and operation codes will be written in octal notation. Also, it will be assumed that address 0000 always stores the constant 0, allows this 0 to be retrieved, but permits no other number except zero to be placed there.

Now we shall describe in brief the instructions used by the hypothetical computer.

**Add: 01abc ADD.** This instruction adds  $(a)$  to  $(b)$ , normalizes the sum and stores it in address  $c$ ;  $\omega = 1$  if  $(c) < 0$ .

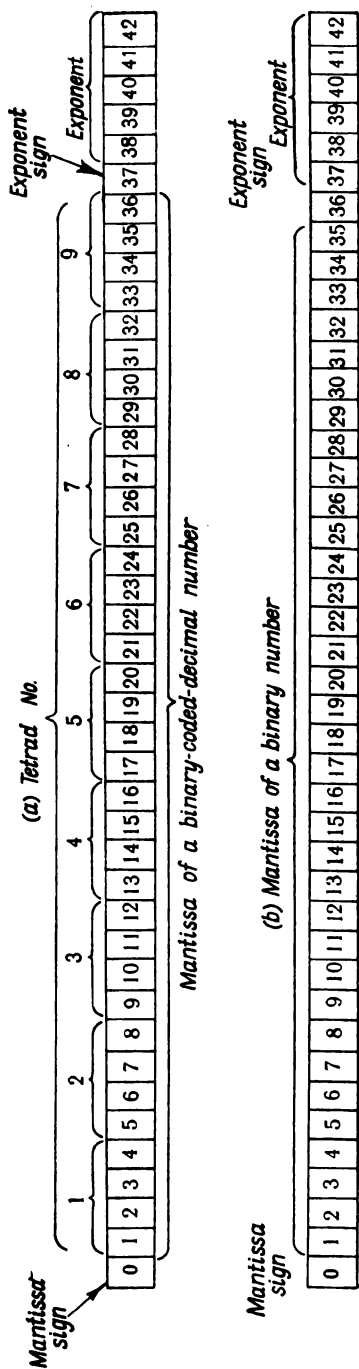


Fig. 8.1

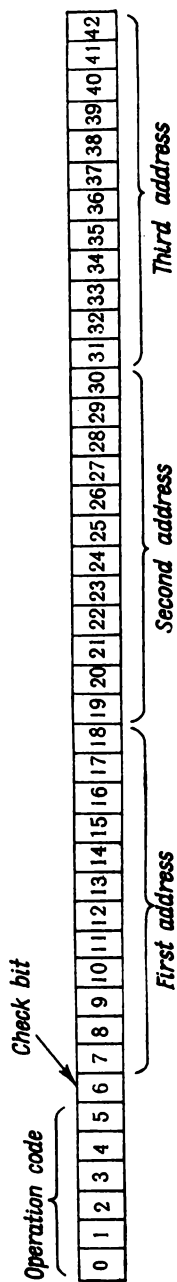


Fig. 8.2



Subtract: *02abc SUB*. This instruction subtracts (*b*) from (*a*), normalizes the difference, and replaces (*c*) with the difference;  $\omega = 1$  if  $(c) < 0$ .

Subtract Magnitude: *03abc SBM*. This instruction subtracts the magnitude of (*b*) from that of (*a*), normalizes the difference, and replaces (*c*) with it;  $\omega = 1$  if  $(c) < 0$ .

Multiply: *04abc MPY*. This instruction multiplies (*a*) by (*b*), normalizes the product, and stores it in address *c*;  $\omega = 1$  if  $(c) < 0$ .

Divide: *05abc DIV*. This instruction divides (*a*) by (*b*), normalizes the dividend, and replaces (*c*) with it;  $\omega = 1$ , if  $(c) < 0$ .

Add Exponent: *06abc ADE*. This instruction replaces (*c*) with the fractional part of (*a*) and the exponent  $\text{EXP}(a) + \text{EXP}(b)$ ,  $\omega = 1$ , if  $(c) < 0$ .

Subtract Exponent: *07abc SBE*. This instruction replaces (*c*) with the fractional part of (*a*) and the exponent  $\text{EXP}(a) - \text{EXP}(b)$ ;  $\omega = 1$ , if  $(c) < 0$ .

Set Sign: *10abc SSN*. This instruction replaces (*c*) with the magnitude of (*a*) and the sign of (*b*);  $\omega = 1$ , if  $(c) < 0$ .

Shift: *11abc SFT*. This instruction shifts (*a*) left if (*b*) is positive and right when it is negative as many places as there are bits in the magnitude of (*b*), and replaces (*c*) with the shifted number;  $\omega = 1$  if all bits of (*c*) are only zeros.

Extract: *12abc EXT*. This instruction carries out columnwise logical multiplication of (*a*) and (*b*), and replaces (*c*) with the logical product;  $\omega = 1$  if all bits of (*c*) are only zeros.

Mask: *13abc MSK*. This instruction carries out columnwise logical addition of (*a*) and (*b*) and replaces (*c*) with the logical sum;  $\omega = 1$  if all bits of (*c*) contain only zeros.

Compare: *14abc CMP*. This instruction takes the columnwise modulo 2 sum of (*a*) and (*b*) and replaces (*c*) with the result;  $\omega = 1$  if at least one bit of (*c*) is nonzero.

Branch: *15abc BCH*. This is a conditional transfer of control instruction; it transfers control according to the value of the  $\omega$  indicator generated by the previous instruction. If  $\omega = 0$ , control is passed to *a*; if  $\omega = 1$ , control is passed to *b*. After BCH is executed, zeros are placed in *c*.

Jump: *15aac JMP*. This is an unconditional transfer of control instruction. It differs from the BRANCH (conditional transfer) instruction in that the second address is replaced with the first address. After JMP is executed, zeros are written in *c*.

The  $\omega$  indicator is not generated anew when BCH, JMP and all subsequent instructions are executed, and the arithmetic unit stores its previous value.

Add Address: *16abc ADA*. This instruction adds *a* to *b* and places the newly formed instruction with the opcode of *a* in *c*.

**Subtract Address: 17abc SBA.** This instruction subtracts  $a$  from  $b$  and places the newly generated instruction with the opcode of  $a$  in  $c$ .

**Tally: 20abc TLY.** This instruction totals the contents of all the 43 bits at  $a$  and  $b$  and performs an end-around carry. The result is placed in  $c$ . The  $\omega$  and  $\phi$  indicators are not generated. This instruction is used for test purposes.

**Copy: 21anc CPY.** This instruction moves  $n + 1$  numbers from locations  $a, a + 1, a + 2, \dots, a + n$  into locations  $c, c + 1, c + 2, \dots, c + n$ , respectively. If  $n = 0$ , only one number will be moved.

**Read In Numbers: 22onc RNO.** This instruction causes the numbers to be read from  $n + 1$  punched cards into locations  $c, c + 1, c + 2, \dots, c + n$ .

**Punch Out Numbers: 23ano PNO.** This instruction causes  $n + 1$  numbers to be punched out of locations  $a, a + 1, a + 2, \dots, a + n$  onto cards.

**Print Out Numbers: 24ano PRN.** This instruction causes  $n + 1$  numbers at addresses  $a, a + 1, a + 2, \dots, a + n$  to be printed as  $n + 1$  lines on paper.

**Convert BCD to Binary: 25anc DB.** This instruction causes  $n + 1$  BCD numbers at addresses  $a, a + 1, a + 2, \dots, a + n$  to be converted to binary form by a standard subroutine or a wired subroutine, into addresses  $c, c + 1, c + 2, \dots, c + n$ , respectively.

**Convert Binary to BCD: 26anc BD.** This instruction causes  $n + 1$  binary numbers at addresses  $a, a + 1, a + 2, \dots, a + n$  to be converted by a stored or wired subroutine into BCD form and placed at addresses  $c, c + 1, c + 2, \dots, c + n$ , respectively.

**Stop Program: 27abc SP.** This instruction stops computer operations and provides a suitable display of (a), (b) and (c) on the control console.

## 8.2. Direct Programming

**8.2.1. Programming procedure.** Programming, that is, setting up a computer algorithm, should begin with arranging the computations involved in a sequence of steps. This sequence depends directly on the numerical methods chosen and the computational equations involved.

As an example, let us evaluate the definite integral

$$I = \int_a^b f(x) dx$$

accurate to a specified  $\epsilon$ . We may well choose numerical integration for our purposes, based on the equation of squares where

$$I = h \sum_{i=1}^n f(x_i)$$

where  $h$  is an iteration step. This last form is a good computational algorithm for the evaluation of a definite integral. Its realization involves the following sequence of steps:

(1) determine the number of iteration steps needed, that is, how many times the instructions of the program must be repeated, and include it in the program as a suitable instruction;

(2) determine  $x_i$ ;

(3) evaluate  $f(x_i)$ ;

(4) evaluate  $f(x_i) + \sum_{i=1}^{i-1} f(x_{i-1})$ ;

(5) go through steps (2), (3) and (4) to evaluate  $\sum_{i=1}^n f(x_i)$ ;

(6) evaluate  $I = h \sum_{i=1}^n f(x_i)$ .

Before the computer can proceed with the computation, the program must be loaded into its storage. This can be done either manually, from the control console, or automatically. Usually, very short programs (not more than 20 to 30 instructions) are loaded manually. With more extensive programs, they are first transferred onto a suitable storage medium (punched cards, say), and program loading is done automatically by a (punched card) reader. Also, before step (2) can be executed, the source data (operands) must be converted to binary notation. After the computer executes the last step, (6), the results must be converted back to decimal notation and printed out or punched into cards. Then the computer stops.

Direct programming is tied up with storage allocation, because space cannot be correctly allocated without knowing how many instructions a program will include, while a program cannot be correctly compiled without knowing the addresses of operands and intermediate results in advance. To avoid this conflict, it is usual to compile a program first in alphameric symbolic addresses. For example, the operands may be allocated locations starting at  $\alpha + 0000$ ; the intermediate (or partial) results, locations starting at  $p + 0000$ ; the initial input program, locations starting at  $d + 0000$ ; and the program instructions, locations starting at  $k + 0000$ .

Once a program has been compiled in terms of symbolic addresses, a clear idea is formed as regards storage requirements for the

program, operands, intermediate and final results, and the program can now be compiled in terms of actual addresses.

Consider a simple example of programming in symbolic and actual (effective) addresses.

**8.2.2. Sequential programming.** We set out to compile a program for our hypothetical computer to evaluate the function

$$y = \frac{Ax^3 - Cx}{D + x^2}$$

for  $x = x_1$ .

To begin with, we allocate storage to the operands  $x_1$ ,  $A$ ,  $C$  and  $D$  at symbolic addresses, beginning at  $\alpha + 0000$ , as shown in Table 8.1.

Table 8.1

$\alpha + 0000$	$\alpha + 0001$	$\alpha + 0002$	$\alpha + 0003$
$x_1$	$A$	$C$	$D$

The locations for partial results will begin at address  $p + 0000$ . The program will be written into locations beginning at address  $k + 0000$ , using mnemonic operation codes.

The first instruction

$k + 0000$  DB  $\alpha + 0000$  0003  $\alpha + 0000$

converts the operands  $x_1$ ,  $A$ ,  $C$  and  $D$  from *BCD* to binary notation and stores the converted values at the previous addresses.

The second instruction

$k + 0001$  MPY  $\alpha + 0000$   $\alpha + 0000$   $p + 0000$

computes  $x_1^2$  and stores the result in partial-result location  $p + 0000$ .

The third instruction

$k + 0002$  MPY  $p + 0000$   $\alpha + 0000$   $p + 0001$

computes  $x_1^3$  and stores the result in partial-result location  $p + 0001$ .

The fourth instruction

$k + 0003$  MPY  $\alpha + 0001$   $p + 0001$   $p + 0001$

computes  $Ax_1^3$  and replaces  $x_1^3$  in partial-result location  $p + 0001$  with the result.

The fifth instruction

$k + 0004$  MPY  $\alpha + 0000$   $\alpha + 0002$   $p + 0002$

computes  $Cx$  and stores the result in partial-result location  $p + 0002$ .

The sixth instruction

$$k + 0005 \text{ SUB } p + 0001 \ p + 0002 \ p + 0001$$

computes the numerator of the fraction and replaces  $Ax_1^3$  in partial-result location  $p + 0001$  with it.

The seventh instruction

$$k + 0006 \text{ ADD } a + 0003 \ p + 0000 \ p + 0000$$

computes the denominator of the fraction and replaces  $x_1^2$  in partial-result location  $p + 0000$  with it.

The eighth instruction

$$k + 0007 \text{ DIV } p + 0001 \ p + 0000 \ p + 0000$$

evaluates the function  $y$  and stores it in location  $p + 0000$ .

The ninth instruction

$$k + 0010 \text{ BD } p + 0000 \ 0000 \ p + 0000$$

converts the value of  $y$  from binary to  $BCD$  form and stores the converted number in location  $p + 0000$ .

The tenth instruction

$$k + 0011 \text{ PRN } p + 0000 \ 0000 \ 0000$$

prints the result on paper.

The eleventh instruction

$$k + 0012 \text{ SP } 0000 \ 0000 \ 0000$$

stops computer operation.

We have thus obtained a program at least in part, in symbolic addresses. Its storage requirements include three partial-result locations at addresses  $p + 0000$ ,  $p + 0001$  and  $p + 0002$  to store the following intermediate and final results:

Table 8.2

$p + 0000$	$p + 0001$	$p + 0002$
$x_1^2, D + x_1^2$ $y_2, y_{10}$	$x_1^3, Ax_1^3$ $Ax_1^3 - Cx$	$Cx$

So that this program can automatically be loaded in computer storage via punched cards, the following initial input program is set up in symbolic addresses:

The first instruction

$d + 0000$  RNO 00000003  $\alpha + 0000$

reads the contents of four punched cards into addresses

$\alpha + 0000$ ,  $\alpha + 0001$ ,  $\alpha + 0002$ ,  $\alpha + 0003$

The second instruction

$d + 0001$  RNO 0000 0012  $k + 0000$

transfers the contents of eleven punched cards (13 cards in octal notation) into locations from  $k + 0000$  to  $k + 0012$  inclusive.

The third instruction of the input program usually describes what the computer should do after data have been read in: halt and wait for the next instruction, or load again and proceed with computation. In the latter case, if the actual address of location  $k + 0000$  is a natural sequence of the actual address of location  $d + 0001$ , the third instruction will normally be omitted. If, on the other hand, the two effective addresses are far removed from each other, the third instruction will be that of unconditional transfer of control to address  $k + 0000$ :

$d + 0002$  JMP  $k + 0000$   $k + 0000$  0000

After a complete program in symbolic addresses has been compiled, the following values will be assigned to the respective letters:  $\alpha = 1000$ ,  $p = 1004$ ,  $d = 0010$ , and  $k = 0100$ . In terms of actual addresses, the program will then appear as shown in Table 8.3.

Now that the program has been represented in terms of actual addresses, the instructions and numbers can be punched onto cards, cards read automatically into the computer, and the computer can proceed with the solution of the problem in accordance with the program loaded.

**8.2.3. Problem preparation for some computers.** With most single-address and three-address computers, programming is fundamentally the same, the main difference being only that the programs for the former are much longer. On the other hand, if the operand of a problem is the result of a previous operation, the number of single-address instructions will be the same as that of three-address instructions. Nor is there any fundamental difference in programming for fixed-point and floating-point machines, although the preparatory steps are usually different.

Fixed-point machines operate on numbers of the form

$$X = xq^m$$

where  $x$  is the fractional or significant part (mantissa) of the number and  $m$  is the exponent, that is, the power of the base  $q$

Table 8.3

Location	Instruction, number	Explanation
0010	22 0000 0003 1000	Read in numbers
0011	22 0000 0012 0100	Read in instructions
0012	15 0100 0100 0000	Exit from initial input routine
.	.	.
0100	25 1000 0003 1000	Convert to binary notation
0101	04 1000 1000 1004	Evaluate $y$
0102	04 1004 1000 1005	
0103	04 1001 1005 1005	
0104	04 1002 1000 1006	
0105	02 1005 1006 1005	
0106	01 1003 1004 1004	
0107	05 1005 1004 1004	Convert to <i>BCD</i> Print out Stop
0110	26 1004 0000 1004	
0111	24 1004 0000 0000	
0112	27 0000 0000 0000	
.	.	.
1000	$x_1$	Operands
1001	$A$	
1002	$C$	
1003	$D$	
1004	0	Partial-result addresses
1005	0	
1006	0	

needed to bring  $x$  up or down to the value of  $X$ , the exponent being constant for all numbers that can be stored in the computer. In most cases,  $m = 0$ , because of which only proper fractions can be stored in a fixed-point machine, and the operands have to be converted so as to bring them down to absolute value less than one (that is, into "digital" number range). Usually, this is done by scaling the operands with the aid of scale factors.

Let the quantity involved in the problem be designated  $x$ . Its scale factor will then be  $M_x$ , defined as the divider such that the scaled quantity,  $\bar{x}$ , is given by

$$\bar{x} = x/M_x$$

and satisfied the inequality

$$|\bar{x}| < 1$$

Obviously,

$$M_x = x/\bar{x}$$

where  $\bar{x}$  is the machine representation of  $x$ . Usually,  $M_x$  is an integer power of 10 (decimal scale factors) or integer power of 2 (binary scale factors).

The use of constant decimal scale factors will be clear from the following example. Suppose we are to compute the value of the function

$$y = \frac{x}{5 - \frac{x^2}{2}}$$

for

$$x_1 = 3.0$$

$$x_2 = 1.5$$

Since the external numbers are greater in absolute value than unity, we choose the scale factor to be  $M_x = 10$ . Then

$$x = M_x \bar{x} = 10\bar{x}$$

On substituting  $x = 10\bar{x}$  in the original equation, we obtain

$$y = \frac{10\bar{x}}{5 - \frac{100\bar{x}^2}{2}} = \frac{0.1\bar{x}}{0.05 - \frac{\bar{x}^2}{2}} = \frac{0.2\bar{x}}{0.1 - \bar{x}^2}$$

The last expression has no coefficients equal to or greater than one, and the variable is likewise less in absolute value than unity ( $\bar{x}_1 = 0.30$ ,  $\bar{x}_2 = 0.15$ ). By the same token, we choose  $M_y = 10$  for  $y$ , and finally write the original equation as

$$\bar{y} = \frac{0.02\bar{x}}{0.1 - \bar{x}^2}$$

By operating on the scaled numbers, the computer will turn out likewise scaled values of the function, that is,  $\bar{y}_1 = 0.6000$  and  $\bar{y}_2 = 0.0385$ . The scaled values can be de-scaled by shifting the decimal point one place right. Then,

$$y_1 = 6.000$$

$$y_2 = 0.385$$

The constant binary scale factors,  $M_x$ , are usually numbers of the form  $2^n$ , that is,  $2^0 = 1$ ,  $2^1 = 2$ ,  $2^2 = 4$ ,  $2^3 = 8$ , etc. Suppose the quantity  $x$  takes on values in the range  $19.5 \leq x \leq 180$  in the course of problem solving and it is required to choose a scale factor,  $M_x$ , as a power of 2. To satisfy this requirement, we



shall find a power of 2 such that dividing the upper bound on  $x$  will give a number  $|\bar{x}| < 1$ . For our example, this will be  $2^8 = 256$ . Applying the scale factor  $M_x = 256$  to the bounds of  $x$ , we obtain

$$\frac{19.5}{256} \leq \bar{x} \leq \frac{180}{256}$$

or

$$0.076 \leq \bar{x} \leq 0.704$$

As is seen, knowledge of bounds on the value of the variable(s) is essential in choosing a correct scale factor. As a rule, finding these bounds is a time and effort consuming job, because of which resort is often made to approximate computations. As often as not, the choice of a scale factor may be based on the physical meaning of the problem.

Scaling based on constant scale factors has the advantages of a simple choice and use of scale factors, but impairs the precision of computation because of the limited number of digit positions allocated to the mantissa. Suppose a computer operates on numbers  $n$  digits long.

As before,  $\bar{x}$  will be the scaled value of the variable  $x$ , and  $M_x$  be the scale factor. On converting  $\bar{x}$  to binary form and reading it into the computer, we obtain an approximate value,  $\bar{x}_{app}$ . Obviously, the absolute error,  $\Delta\bar{x}$ , will be less than  $1 \times k^{-n}$ . Multiplying  $\Delta\bar{x}$  by  $M_x$  gives

$$\Delta x = k^{-n} M_x$$

As is seen,  $\Delta x$  varies directly as  $M_x$  (Fig. 8.3). This is why the fixed scale factor method should preferably be limited to cases where the precision required is low. Where precision is important, resort should be made to the floating scale factor method by which scale factors are only selected for external numbers, while those of the machine variables are changed automatically by a special routine loaded into the computer.

**8.2.4. Branching programs and branch instructions.** Normally (as has been shown above), a computer executes the list of instructions sequentially, that is, in the same order as they have been written into the program. Under some circumstances, however, it may be necessary to depart from the sequential order and go to an entirely different part of the program. The process of computation is then said to branch, and the instructions causing

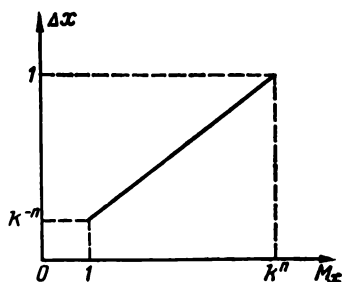


Fig. 8.3

this to happen are called branch (or conditional transfer) instructions.

For a program to branch in any one of two possible directions, two instructions are needed. One instruction tests if some specified condition has been satisfied and generates an " $\omega$ " signal; the other instruction performs the transition to another part of the program according to the value of  $\omega$  generated by the first instruction. If the program can branch into more than two directions, the condition must be tested for each branch.

**Example.** Write a program for the hypothetical computer to evaluate the functions  $f_1(y)$ ,  $f_2(y)$  and  $f_3(y)$  for specified values of  $x$ , if

$$y = x^3 + x - A \quad (1)$$

where

$$A > B > 0$$

such that

$$\text{for } |y| \leq A, \quad f_1(y) = y^2 - A \quad (2)$$

$$\text{for } A < |y| \leq B, \quad f_2(y) = By - A \quad (3)$$

$$\text{for } |y| > B, \quad f_3(y) = Cy^3 - A \quad (4)$$

The process of computation has three branches which can be effected as two consecutive branches in two directions. As the first step, we evaluate  $y$  and test the condition

$$A - |y| < 0$$

If the condition is not satisfied, that is, if

$$|y| \leq A$$

we shall use Eq. (2); otherwise, we shall test the condition

$$B - |y| < 0$$

If this condition is not satisfied, that is, if

$$|y| \leq B$$

we shall determine  $f_2(y)$ ; otherwise Eq. (4) is used. Omitting the initial input routine, number conversion, result display and machine stop, we assign storage to the operands (Table 8.4) and select partial-result addresses (Table 8.5). The resultant program in symbolic addresses appears in Table 8.6. Graphically, the program may be presented as shown in Fig. 8.4. In fact, the method described in the above example may be used to set up programs for any number of branches.

**8.2.5. Computation loops and loop instructions.** In many cases, computations by means of numerical methods reduce to the repetitive use of the same formulae. When a cycling process of this type is taking place, the computer is said to be in a loop of the

Table 8.4

$\alpha + 0000$	$\alpha + 0001$	$\alpha + 0002$	$\alpha + 0003$
$x$	$A$	$B$	$C$

Table 8.5

$p + 0000$	$p + 0001$	$p + 0002$
$x^2, x^3, x^3 + x, y$	$y^2, By, y^3$	$f_1(y)$

program, and each traverse of the loop is called a *cycle*. Obviously, for a computer to enter or leave a loop at the correct stage in a computation, it must be instructed appropriately. This is done by setting up a loop (or, rather, iterative loop) program. An iterative

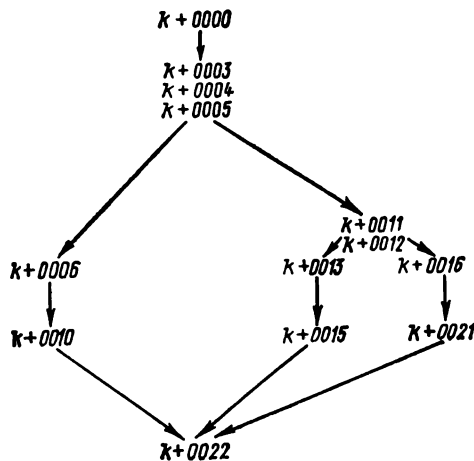


Fig. 8.4

loop program enables a computer to execute a great number of arithmetic and logical operations in response to a small number of instructions.

As an example, we consider a simple iteration loop. Suppose there is a number,  $x$ , stored at address  $\alpha + 0000$  and it is necessary to find the value of  $y = x^{15}$  and store it in location  $p + 0000$ . Of course, the problem can readily be solved by a complete (linear) program like that shown in Table 8.7.

Table 8.6

Location	Instruction	Explanation
$k + 0000$	MPY $\alpha + 0000$ $\alpha + 0000$ $p + 0000$	$x^2$
$k + 0001$	MPY $p + 0000$ $\alpha + 0000$ $p + 0000$	$x^3$
$k + 0002$	ADD $p + 0000$ $\alpha + 0000$ $p + 0000$	$x^3 + x$
$k + 0003$	SUB $p + 0000$ $\alpha + 0001$ $p + 0000$	$x^3 + x - A = y$
$k + 0004$	SBM $\alpha + 0001$ $p + 0000$ 0000	$A -  y $ ; if $A -  y  \geq 0$ , then $\omega = 0$ ; if $A -  y  < 0$ , then $\omega = 1$
$k + 0005$	BCH $k + 0006$ $k + 0011$ 0000	If $\omega = 1$ , control is passed to $k + 0011$ ; if $\omega = 0$ , to $k + 0006$
$k + 0006$	MPY $p + 0000$ $p + 0000$ $p + 0000$	$y^2$
$k + 0007$	SUB $p + 0000$ $\alpha + 0001$ $p + 0002$	$f_1(y)$
$k + 0010$	JMP $k + n$ $k + n$ 0000	Control is passed to closing instructions
$k + 0011$	SBM $\alpha + 0002$ $p + 0000$ 0000	$B -  y $ ; if $B -  y  \geq 0$ , then $\omega = 0$ ; if $B -  y  < 0$ , then $\omega = 1$
$k + 0012$	BCH $k + 0013$ $k + 0016$ 0000	If $\omega = 1$ , control is passed to $k + 0016$ ; if $\omega = 0$ , to $k + 0013$
$k + 0013$	MPY $\alpha + 0002$ $p + 0000$ $p + 0001$	$B y$
$k + 0014$	SUB $p + 0001$ $\alpha + 0001$ $p + 0002$	$f_2(y)$
$k + 0015$	JMP $k + n$ $k + n$ 0000	Control is passed to closing instructions
$k + 0016$	MPY $p + 0000$ $p + 0000$ $p + 0001$	$y^2$
$k + 0017$	MPY $p + 0001$ $\alpha + 0003$ $p + 0001$	$C y^2$
$k + 0020$	MPY $p + 0001$ $p + 0000$ $p + 0001$	$C y^3$
$k + 0021$	SUB $p + 0001$ $\alpha + 0001$ $p + 0002$	$f_3(y)$
$k + 0022$	JMP $k + n$ $k + n$ 0000	Control is passed to closing instructions
$\vdots$		
$\vdots$		
$k + n$	First of closing instructions	

Table 8.7

Location	Instruction	Result
$k + 0000$	MPY $\alpha + 0000$ $\alpha + 0000$ $p + 0000$	$x^2$
$k + 0001$	MPY $\alpha + 0000$ $p + 0000$ $p + 0000$	$x^3$
$k + 0002$	MPY $\alpha + 0000$ $p + 0000$ $p + 0000$	$x^4$
	$\vdots$	$\vdots$
	$\vdots$	$\vdots$
	$\vdots$	$\vdots$
0014	MPY $\alpha + 0000$ $p + 0000$ $p + 0000$	$x^{14}$
0015	MPY $\alpha + 0000$ $p + 0000$ $p + 0000$	$x^{15}$

14 instructions

A simpler way to solve the problem is to use an iterative loop program. We write "1" in location  $\alpha + 0001$ , "13" in location  $\alpha + 0002$ , and  $x$  in location  $\alpha + 0000$  as before, and set up a program as shown in Table 8.8.

Table 8.8

Location	Instruction	cycle I	cycle II	. . .	cycle XIII
$k + 0000$	MPY $\alpha + 0000$ $\alpha + 0000$ $p + 0000$	$x^2$			
$k + 0001$	MPY $\alpha + 0000$ $p + 0000$ $p + 0000$	$x^3$	$x^4$	. . .	$x^{15}$
$k + 0002$	ADD $\alpha + 0001$ $p + 0001$ $p + 0001$	1	2	. . .	13
$k + 0003$	SUB $p + 0001$ $\alpha + 0002$ 0000	$1 - 13 =$ $= -12$	$2 - 13 =$ $= -11$	. . .	$13 - 13 =$ $= 0$
$k + 0004$	BCH $k + 0005$ $k + 0001$ 0000	BCH on $\omega = 1$	BCH on $\omega = 1$	. . .	BCH on $\omega = 0$

In the above program, location  $p + 0001$  is called a cycle counter; prior to the start of a computation the cycle counter is cleared to zero. Each time the iteration loop is traversed, the contents of the cycle counter are augmented by 1, and the number of cycles traversed is subtracted from the preset number (13 in location  $\alpha + 0002$ ). As long as the accumulated number of cycles in location  $p + 0001$  is less than 13, the instruction at address  $k + 0003$  generates a signal,  $\omega = 1$ , and the next conditional jump instruction passes control to location  $k + 0001$ , thereby causing the loop of instructions to be obeyed again. At the end of the 13th cycle,  $\omega = 0$ , and the BCH instruction causes another route to be chosen (say, to location  $k + 0005$ ). Thus, with an iteration loop procedure, five instructions are needed instead of 14, as with the linear or sequential technique.

In the above example, the number of cycles round the loop of instructions was known in advance. This is not so in many cases, however, such as with successive approximation procedures. Rather, it has to be determined by the computer itself in the course of computation on the basis of partial results.

In solving a problem by the successive approximation method, the computation begins by guessing a value for  $x$ , using the formula  $x_{i+1} = f(x_i)$ . Then the difference  $x_{i+1} - x_i$  is tested to see if it satisfies the required accuracy. If it does not, an improved value is chosen for  $x$ , and the procedure is repeated until the inequality

$$|x_{i+1} - x_i| - \epsilon \leq 0$$

is satisfied.

In conclusion, it is worth while examining some methods of loop control.

(1) If the number of cycles round the loop of instructions is not known in advance, loop control is effected by applying a test for, say, a logical condition of the form

$$|x_{i+1} - x_i| - \epsilon < 0$$

and causing the loop to be traversed until the monotonically decreasing quantity becomes less than predetermined value.

In the general case, this form of loop control may be realized as follows. Let  $\alpha + 0000$  be a monotonically decreasing quantity, and  $\alpha + 0001$  be the preset minimum value. Then

```

→ k + 0000
  .
  .
  .
  k + m      SUB α + 0000 α + 0001 0000
└─ k + m + 1 BCH k + 0000 k + m + 2 0000
→ k + m + 2

```

(2) The loop of instructions may be caused to be traversed until a monotonically increasing quantity exceeds a predetermined value. Let the monotonically increasing quantity be stored in location  $\alpha + 0000$ , and the predetermined maximum value in location  $\alpha + 0001$ . Then, in the general case, loop control can be programmed as follows:

```

→ k + 0000
  .
  .
  .
  k + m      SUB α + 0001 α + 0000 0000
└─ k + m + 1 BCH k + 0000 k + m + 2 0000
→ k + m + 2

```

(3) When the number of cycles round the loop of instructions is known in advance, loop control can best be organized, using a cycle counter as described earlier. If some instruction can be arranged to change with each loop traverse so that its final form is known, the cycle counter method can be used in a somewhat modified form. The final form of the modifiable instruction is written in symbolic form in the counter and the current form is compared against it. The result of the comparison is then utilized by the conditional jump instruction to cause the loop to be obeyed or not.

**8.2.6. Use of standard subroutines in programming.** Many problems run on a computer involve evaluating a square root, taking a natural logarithm, exponentiation, or evaluating trigonometric functions, and also converting operands and results from binary to *BCD* notation and back. These are all standard operations, and it was early realized that they could be tackled efficiently by pre-coded and pretested programs known to work correctly. Such programs are called standard subroutines, usually arranged into libraries. Libraries of standard subroutines differ widely in both composition and size.

A subroutine is constructed so that it has fixed locations for operands and results, with addresses assigned in advance. Before a subroutine is called in, the arguments on which it is to operate are stored in its standard locations. At the end of a subroutine there is a vacant location, an exit location, where an unconditional jump instruction is written (before the subroutine is called in), to transfer control to an appropriate address in the main program. In turn, the main program incorporates an instruction which causes the standard subroutine to be accessed at its first (initial) address.

A library of standard subroutines may be stored in internal or external memory. In the latter case, more frequently used because external storage has a far greater capacity, the subroutines are transferred into uniquely defined locations of internal storage before a computation begins. Subroutines left at relative addresses may be written on standard forms and then transferred to actual addresses when they are included in the main program. However, this involves the expenditure of an additional time. The latest trend has been towards storing standard subroutines at relative addresses which are copied and modified by the computer itself, using so-called compiling or merging subroutines.

### 8.3. An Outline of FORTRAN Language

**8.3.1. General.** There are quite a number of methods which attempt to shift the burden of programming of problems for computers from the human programmer onto the machines themselves.

ves. Quite appropriately, this is referred to as automatic programming and includes in part, subroutine libraries, symbolic addressing, string language programming, programming programs, and address programming.

Automatic programming is especially efficient when it makes use of special algorithmic or programming languages. The characters and symbols used by algorithmic languages bear a very close resemblance to the usual mathematical notation, and it turns out far simpler for the human operator to learn such languages than programming under the constraints imposed by the particular computer he happens to be using. Also, programs written in algorithmic languages are easy to grasp and help to avoid any ambiguities in reading.

Because an algorithmic language uses a limited number of characters and symbols and also because a program written in an algorithmic language (the source program) contains all the information needed about the path of control, the source program can readily be translated into a machine (object) program by the computer itself, if it is provided with a special programming program called a translation program, or simply a *translator*. The alternative names for it are a compiling program or a *compiler*. As a result, an operator with a working knowledge of an algorithmic language can set up problems for any digital computer provided with a translation program.

One such language, widely used at present, is the FORTRAN language, specifically designed for scientific and engineering applications.

FORTAN is an acronym for FORMula TRANslation. However, it has proved to be more versatile than that. It can readily be used to describe problems involving advanced logic, simulation problems, editing, and a variety of business applications.

Basic FORTRAN dates back to 1954. Since then, several versions have appeared, and work continues.

**8.3.2. Character set of FORTRAN language.** Programs in FORTRAN are written, using the following characters:

Alphabetic:	A through Z
Numeric:	0 through 9
Special:	
	Space
+	Plus
-	Minus
*	Asterisk (multiplication sign)
/	Slash (division sign)
=	Equals
(	Left parenthesis



)	Right parenthesis
,	Comma
.	Decimal point
"	Quotation mark

Spaces are used as data item delimiters. In Soviet-made machines, Russian characters may also be used.

The decimal point separates the fractional part of a number from its integral part. In FORTRAN, the "=" (equals) sign marks the operation of assignment; it causes the constant or expression written on the right of the "=" sign to be assigned to the variable written on its left.

**Constants.** A constant is a value that is always defined during program execution and may not be re-defined. Two types of constants are used in FORTRAN: integer and real.

Integer constants are integers without a decimal point, for example:

-03 +9 18 -6 -243

Real constants may be expressed:

- (a) as an integer part with a decimal point;
- (b) as a decimal fraction with a decimal point in front of the fractional part;
- (c) as a mixed number in which the integral part is followed by a decimal point.

A constant may take a "+" or a "-" sign. An unsigned constant is recognized as positive.

A real constant may include an exponent, in which case it consists of two parts, the constant proper and its exponent (an integer power of 10). The exponent is written as an integer preceded by the letter E. The exponent symbol may be followed by a "+" or a "-" sign. An unsigned exponent is recognized as positive. Examples of real constants are:

1.      0.2    -.0097   6.0  
          5.0E3 5.0E - 3 5.0E + 3

The last three constants are equivalents of:

$5.0 \times 10^3$ ,  $5.0 \times 10^{-3}$ , and  $5.0 \times 10^3$ , respectively.

**Variables.** A variable in FORTRAN is identified by a symbolic name. Integer (fixed-point) and real (floating-point) variables are recognized. The type of variable is identified by the first letter of the symbolic name. The letters I, J, K, L, M, and N indicate an integer-(fixed-point) variable; any other letter indicates a real (floating-point) variable.

Examples of symbolic names are:

I            M3 MAX    KEY1

Integer variables may take on any values having the form of an integer constant. For example, if *I* is an integer variable and 3 is an integer constant, *I* can take on a value equal to 3, but it cannot be assigned a value of 3.25, because this is a real constant.

Real variables are identified by alphanumeric characters (except special characters); the first character must be alphabetic, except the letters I, J, K, L, M, and N. Examples of real variables are.

#### A B22 DELTA PRICE

Real variables can take on values equal to any real constant. If an operation on integer variables and constants produces a fraction, the fraction is truncated, that is, the digits following the decimal point are discarded without rounding-off.

**Examples:**

<i>Arithmetic operation</i>	<i>Result</i>
$I = 5/2$	$I = 2$ (Instead of 2.5)
$K = 5/2 + 7/2$	$K = 5$

**Subscripts.** Subscripts may be one of the following forms:

$$\begin{aligned}
 &V \\
 &C \\
 &V + C \text{ or } V - C \\
 &C * V \\
 &C * V + C1 \text{ or } C * V - C1
 \end{aligned}$$

where *V* is an unsigned simple integer variable, and *C* and *C1* are any unsigned integer constants.

Real variables and constants cannot be used as subscripts. Examples of valid subscripts are:

$$4 * M3 \text{ IMA}X \text{ 198} * \text{QUANTUM}$$

A subscripted variable may be an integer or real variable used as an identifier, with one or two subscripts separated by commas and enclosed in parentheses. Examples of valid subscripted variables are:

$$\begin{aligned}
 &A(I) \qquad \qquad K(3) \\
 &BETA(5 * J - 2, K + 2, L) \\
 &MAX(J, 2)
 \end{aligned}$$

A subscripted variable defines an element of an array. The array and its dimensions must be declared at the very first appearance of the subscripted variable, in a DIMENSION statement.

Generally, a DIMENSION statement is of the form

$$\text{DIMENSION } V1, V2, \dots, Vn$$

where  $V$  is an array declarator which may be a variable with one, two or three subscripts. The subscripts are unsigned integer constants. All subscripted variables must be separated by commas:

DIMENSION NUMBER (10), A (10), B (5, 15)

For the first time, a subscripted variable can only appear in a DIMENSION statement. This statement specifies the maximum array length. This point must not be overlooked in an object program, or else the specified limits might be exceeded in referring to a given array. For example, the statement

DIMENSION NUMBER (10, 10, 10)

defines a three-dimensional array whose name is NUMBER and whose subscripts for each dimension cannot exceed 10. Also, the DIMENSION statement indicates that the array whose name is NUMBER will occupy 1000 storage locations ( $10 \times 10 \times 10$ ). One DIMENSION statement may reserve storage for any number of arrays, while these arrays may be integer or real, or both.

**Number range and precision.** For the most part, FORTRAN programs operate on numbers having seven significant digits, which usually ensures a sufficient precision. Yet, there may be cases where a higher precision is important. In such cases, the double precision (D) form is used for real variables and constants, and the fractional part (mantissa) of the number may have up to 16 significant digits.

**8.3.3. Arithmetic statements.** An arithmetic statement is not unlike an arithmetic expression and defines the computation to be executed. Generally, an arithmetic statement is of the form

$$A = B$$

where  $A$  is a variable (simple or subscripted) and  $B$  is an arithmetic expression. Examples of arithmetic statements are:

$$A = B + C$$

$$D(I) = E(I) \cdot 2. - F$$

An arithmetic expression may be a constant, a simple or subscripted variable, or a function. Arithmetic expressions may be combined by arithmetic operators and parentheses to form complex arithmetic expressions. The following five arithmetic operators are used:

+ Addition

— Subtraction

• Multiplication

/ Division

•• Exponentiation

The “\*\*” operator occupies two positions in a coding form, but it is recognized as the sign of a single operation.

Since constants, simple and subscripted variables may be integer or real, arithmetic expressions may contain both integer and real quantities, but both types can only appear in an expression in accordance with definite rules.

(1) A simple arithmetic expression may be a constant, a simple or a subscripted variable. If the constituent element is an integer, the expression is said to be integer. If, on the other hand, the constituent element is real (that is, it has both an integer and a fractional part), the expression is said to be real.

Examples of integer expressions are:

3    I   I(J)

Examples of real expressions are:

3.0   A   A(J)

The expression A(J) must have an integer subscript, but this does not affect the type of expression; the type of expression is decided solely by the constituent element proper.

(2) The exponent of the element that forms an expression does not affect the type of expression. However, it is only a real quantity that can be exponentiated by a real operand. Examples of valid exponential expressions are:

I \*\* I   (integer)

A \*\* I   (real)

A \*\* B   (real)

(3) Quantities preceded by a “+” or a “—” sign and combined by any of the arithmetic operators will form a valid arithmetic expression, subject to the following conditions:

(a) an arithmetic expression may not contain adjoining arithmetic operators;

(b) the operands of an arithmetic expression must be all of the same type (integer or real), except that real variables and constants may be exponentiated by integer operands;

(c) no arithmetic operator may be omitted or assumed.

(4) Parentheses in a arithmetic expression do not affect its type. That is, A, (A) and ((A)) are all real expressions.

(5) Parentheses may be used to indicate the hierarchy of arithmetic operations. If there are no parentheses, the hierarchy of

arithmetic operations is:

**	exponentiation	class 1
/	division	
*	multiplication	class 2
—	subtraction	
+	addition	class 3

The expression on the right of the equals sign may be of a different type than that on the left. If the expression on the left is integer and that on the right is real, the former is evaluated as a real expression, to yield a truncated result: the digits following the decimal point are discarded to leave the integer part alone, without any rounding-off. For example, if the result is  $+3.872$ , it will be truncated to  $+3$ , and not rounded off to  $+4$ . If the expression on the left of the equals sign is real and that on the right is integer, the result will be a real quantity.

**8.3.4. Control statements.** Program execution normally proceeds from statement to statement as they appear in the program. This normal sequence can be altered by means of control statements which include GO TO, IF and DO statements.

**GO TO statements.** These statements provide an unconditional transfer of control to a labelled statement. The general form of this statement is:

GO TO n

where n labels the statement to be executed after the GO TO statement.

**IF statements.** The arithmetic IF statement provides a conditional transfer of control. The general form of this statement is:

IF (A) n1, n2, n3

where A is any arithmetic expression, and the n's (n1, n2, n3) are statement labels. The expression A must be enclosed in parentheses. The statement labels are separated by commas.

**Example:**

IF (A (I)/D) 1, 8, 3

Control will be passed to the statement with label n1, n2 or n3, according to whether the expression A is less than, equal to or greater than zero, respectively.

**Loops.** FORTRAN offers several ways for specifying a loop. One is by means of logical IF statements. A simple example of a loop is the evaluation of the factorial of a number N:

$$N! = N * (N - 1) * (N - 2) * \dots * 2 * 1$$

or, in an "unfolded" form:

```

      IFACT = 1
      K = N
1      IFACT = IFACT * K
      K = K - 1
      IF (K) 2, 2, 1
2      Continue program

```

In the above example, the result  $N!$  is assigned to a variable, IFACT. The first two statements of the program precede the loop and specify the original states of the variables IFACT and K. The statement  $IFACT = 1$  gets the variable IFACT ready for the first multiplication.

The statement  $K = N$  is essential, if the quantity  $N$  is to be preserved in the course of computation. The statements

```

1      IFACT = IFACT * K
      K = K - 1
      IF (K) 2, 2, 1

```

constitute a loop proper. The first digit in the subscript of the IF statement has no sense here, because  $K$  will never be negative. The IF statement causes the computer to repeat this group of statements  $N$  times until  $K$  is reduced to zero. The  $K = K - 1$  statement decrements the variable  $K$  from its original state,  $N$ , by one each time the loop is traversed. The statement

```

      IFACT = IFACT * K

```

accumulates in the variable IFACT the products which lead to  $IFACT = N!$  after the loop has been traversed  $N$  times.

**DO statements.** The previous example has illustrated the use of an IF statement for loop control through a count parameter which is incremented with each traverse of the loop. This involves three statements, a preparatory statement to specify the initial value of the counter, a statement to increment the count, and an IF statement to control the loop.

FORTRAN offers a special means for loop programming with which the above three statements can be replaced with a single one. This is the DO statement which has the form:

```

      DO n1 = m1, m2, m3

```

where

$n$  = the label of an executable statement which ends the group of statements

$i$  = control variable which may be a simple integer variable  
 $m$ 's = indexing parameters which may be unsigned integer constants or simple integer variables

The terms on the right of the equals sign must be separated by commas. The simple integer variable on the left of the equals sign in the specification of a loop statement is called a loop index.

A DO statement makes it possible to repeat a specified number of times the group of statements following it, to and including the terminal statement labelled  $n$ ,  $m_1$  is the initial value of the control variable  $i$  (in the first traverse),  $m_2$  is the terminal value of  $i$  (in the last traverse), and  $m_3$  is the incrementation parameter (that is, the value by which  $i$  is incremented each traverse). If  $m_3$  does not appear, the incrementation value is assumed to be 1.

Thus, a DO statement causes the following steps to occur at the time of execution:

(1) The control variable is assigned the value of the initial parameter.

(2) The range of the DO is executed the specified number of times.

(3) The control variable is increased by the value of the incrementation parameter specified by the programmer on each traverse of the loop.

In the example that follows, the DO statement defines the loop as a group of statements labelled 7, 3 and 2. It specifies the loop to be traversed 25 times (the number of traverses is equal to the difference between the second and first indexing parameters incremented by one):

```

DO2J = 1,25
7      K(J) = J * 2
3      L(J) = J * 3
2      M(J) = L(J) - K(J)

```

After the DO loop is satisfied, control transfers to the statement following  $n = 2$ .

Within a DO loop, any statement may be specified, while there are certain limitations on the terminal statement (its label  $n$  is specified in the DO statement): it must physically follow the DO statement in the source program and it may not be a GO TO of any form or IF statement. If a DO loop is to be terminated with transfer of control, the terminal statement must be followed by one more statement, CONTINUE. This is a no-operation statement most frequently used to provide a loop termination and its label must be specified in the DO statement. Without a DO statement, a CONTINUE statement acts as a do-nothing instruction.

As an example, suppose we are to find the first element equal to ten in the  $X(I)$  array. This involves two requirements: firstly, if  $X(I) \neq 10$ , the computer must be instructed to proceed with checking the next element and, secondly, if all array elements turn out to be not equal to 10, an exit from the loop must be provided. The solution is offered by the following statements:

DO 5 I = 1, 200

IF (X (I) - 10) 5, 10, 5

5 CONTINUE

A CONTINUE statement is required for every loop, although as such it does not affect or use array elements. Each "execution" of the CONTINUE statement marks the completion of an iteration, that is, it causes the control variable to be increased by the incrementation value, the sequence to be repeated starting at the statement that immediately follows the DO if the control variable is less than the terminal parameter, or a transfer out of the loop to take place if the control variable has exceeded the terminal parameter.

A normal transfer out of a loop takes place when the control variable is equal to (or falls shorter than the incrementation value of) the terminal parameter after the terminal statement of the loop has been executed. If this occurs when the control variable is less than the terminal parameter, a special transfer out of the loop has taken place.

A DO loop is an independent member of a program. Yet, there are certain rules which govern relationships between the statements inside and outside of a DO loop.

(1) It is illegal for a GO TO or IF statement to initiate a transfer of control from outside into the range of a DO. Transfer outside of the loop may only be made to the start of the range, that is, to the DO statement itself. An IF or GO TO statement can transfer control only to statements within the same loop.

(2) The range of a DO loop may contain one or more other DO loops (in which case the latter are said to be nested). However, DO loops may be nested as long as their ranges do not overlap; that is, an inner DO must be wholly inside an outer DO.

**STOP statement.** The STOP statement terminates the execution of the program so that it can only be resumed from its very beginning. If the RUN button is pressed, the halt operation is repeated.

**PAUSE statement.** In contrast to the STOP statement, the PAUSE statement provides a temporary program halt; that is, if the RUN button is pressed, the execution of the program will



be resumed at the first statement following the PAUSE statement.

**END statement.** This is a nonexecutable statement. Its function is to indicate the physical end of a program to the translator. The END statement is required for every program. Its place is precisely specified — the END statement is always the last statement of a program.

**8.3.5. Main Input-Output Statements.** For the most part, I/O statements perform three functions:

- (1) They specify what is to be done: to read punched cards, print a line, read magnetic tape, write on magnetic tape, etc.
- (2) They define how data shall be arranged for input/output.
- (3) They specify the data to be transferred between memory and an external unit.

**READ statement.** This statement causes data to be read from one punched card:

READ n, list

where

n = label of a FORMAT statement (see below)

list = list of variables. The variables in a list may be integer, real or both

For example,

READ n, I, J, A, B, C

is a valid statement. The list of the READ statement may include subscripted variables. The subscripts, if any are used, may be constants or variables whose values are defined at the time of execution of the READ statement. For example,

READ n, A, X(3), B, L(J)

will read numbers from an input unit and assign their values to the variable A, the third element of the X array, the variable B, and to the Jth element of the L array (J must be defined by the time of execution of the READ statement).

If the list of variables of the READ statement includes a subscripted variable, the subscript need not be present in the same list: it may be defined elsewhere by any previous statement. In the example that follows, a DO statement defines the subscript of a variable in the READ statement:

DO 5 I=1,8

5 READ 2, X(I)

The DO statement will cause the READ statement to be repeated eight times, each time with a new value of I. In this way, the eight values read by the READ statement are assigned to the elements of the X(I) list.

The READ statement offers a means to input lists and array tables.

**FORMAT statement.** The data to be transferred between an external unit and memory are defined by a FORMAT statement. This is a nonexecutable statement; it only supplies I/O statements with information on the format of data to be input and results to be output. The FORMAT statement must be labelled.

A FORMAT statement can specify the number of data items punched on a card, their presentation, and the number of significant digits or digits following the point.

For example,

3 FORMAT (5F8.5)

READ 3, A, B, C, D, E

indicates that execution of the READ statement will cause five octal numbers (A, B, C, D, E) to be read from the same punched card, and a point to be inserted to separate a three-digit integer part.

A FORMAT statement consists of a keyword, FORMAT, and two parentheses enclosing information on data disposition on a storage medium. This information is given by an alphameric code. The alphabetic characters of this code define where fixed- or floating-point presentation is used. Integer numbers are defined by the letter I, while the letter F in a FORMAT statement will define a fixed-point real number and the letter E, a floating-point real number.

**Example:**

1 FORMAT (14I5)

defines fourteen integer numbers; the parentheses enclose the letter I followed by the integer 5 which defines the area length (the number of characters in each number).

**Example:**

1 FORMAT (7F10.4)

defines seven fixed-point numbers each allocated ten digit positions of which four are occupied by the fractional part. The second field after F is the position identifier for the decimal point and must be included in every FORMAT statement. However, the computer will recognize this identifier only if no decimal point has been punched into the card, that is, no point is defined in the number record. In such a case, its position is specified by the FORMAT statement. A decimal point punched into a card has priority over the decimal point identifier of the FORMAT statement; in such a case, the computer will only take cognizance of the area length, and the decimal point will be marked as pun-

ched. With a punched decimal point, the area length includes the decimal-point column.

A statement of the form

**1 FORMAT (5E13.5)**

defines five floating-point numbers, each with an area 13 digit positions long, of which five digits are in positions following the decimal point.

If the number on the card being read is 12345.6E-2 (including the decimal point), while the FORMAT statement defines it as E10.2, it will be read as 123.456 on input into the computer.

**WRITE statement.** This is a universal output statement. It consists of a keyword, WRITE, the FORMAT statement label, and a list of names of output variables and constants.

**Example:**

**WRITE 3, A**

will cause the value of A to be transferred to a printer in the format defined by 3 FORMAT.

The WRITE statement refers to the FORMAT statement in order to specify the form of output. In itself, a WRITE statement lists the quantities in the order in which they are to be transferred out of the computer to an external storage medium. The rules by which the list of output variables is formed are basically the same as those for the list of names of variables for READ statements: the same use of commas, parentheses and any forms of variables.

For example, a statement of the form

**WRITE 2, J, I, K, A, X(3)**

will cause a printer to print three numbers in the I form, one real number (whether it is to be in the F or E form can only be defined by FORMAT statement No. 2), and a real number, X(3) of array X.

Results are transferred to a printer line-by-line. The form of each line is specified by the FORMAT statement. Statements of the form

**5 FORMAT (F4.2, E6.2, I4)**

**WRITE 5, A, B, L**

will cause two real numbers in the F and E form and one integer number to be printed.

A FORMAT statement allows several numbers to be printed in the same line, with spaces between them. Two statements of

the form

3 FORMAT (F10.2, 10X, E6.2)

WRITE 3, A, B

will cause two real fixed- and floating-point numbers to be printed, separated by 10 spaces.

The maximum number of characters per line depends on the type of printer used by the computer. The computer will print as many characters as is specified by the FORMAT statement. A decimal point is printed always. Its position is defined by the F and E codes of the FORMAT statement.

In specifying the E code and the number of characters to be printed, it should be remembered that in addition to the characters of the fractional part (mantissa), room should be reserved for the following characters:

- (1) a zero in the units position of the mantissa;
- (2) the decimal point of the mantissa;
- (3) the E character;
- (4) one position preceding the zero in the integral part of the mantissa; this position is reserved for the sign of the number and may be left blank in the case of a positive number;
- (5) one position immediately following the E character; this position is reserved for the sign of the exponent and may be left blank for numbers greater than one.

Also, it should be remembered that the exponent is always printed as a two-digit number; exponents less than 10 are written as two digits the first of which is zero.

Thus, in order to construct the first component of the E code, at least 7 must be added to the desired number of significant digits, if the loss of the most significant digits is to be avoided.

**Input/output of alphabetic information.** For any text to be printed, it will suffice to write this text in quotation marks inside the parentheses of a FORMAT statement. For example,

2 FORMAT ('FORTRAN')

WRITE 2

will cause the word FORTRAN (less quotation marks) to be printed.

**8.3.6. Subprograms.** FORTRAN offers the choice of four types of subprograms as follows:

- (1) Library subprograms.
- (2) Arithmetic subprograms.
- (3) FUNCTION subprograms.
- (4) SUBROUTINE subprograms.

**Library subprograms.** Each library subprogram has a unique name. In order to call a subprogram from the FORTRAN library,

it will suffice to reference it by its name in much the same manner as variables are specified. If we seek to evaluate, say, the sine of  $X$ , we shall simply write

$$Y = \text{SIN}(X)$$

where SIN is the name of the sine library subprogram. The argument of the subprogram must be enclosed in parentheses immediately following the subprogram name. After the execution of the subprogram, the value of the sine is assigned to  $Y$ .

A list of basic subprograms which are part of the FORTRAN library follows.

Symbolic name	Description
SIN (X)	Trigonometric sine
COS (X)	Trigonometric cosine
TANH (X)	Hyperbolic tangent
(the arguments of the above functions must be specified in radians)	
ATAN	Arctangent
SQRT (X)	Square root
EXP (X)	Exponential
ALOG (X)	Natural logarithm
ALOG10 (X)	Decimal logarithm

The subprograms have real arguments, and the results of all of these subprograms are likewise real.

The FORTRAN library is not limited to the subprograms listed above. Other library subprograms may use integer arguments as well. For example, the IABS(M) subprogram forms the absolute value of the argument  $M$ .

The name of a subprogram is an alphameric identifier of up to six characters. The first character must always be alphabetic; it defines the form in which the subprogram presents its results.

**Arithmetic subprograms.** These subprograms execute arithmetic statements and are not included in the FORTRAN library. An arithmetic subprogram may use different arguments each time it is being executed. With the definition of an arithmetic subprogram written only once, it may then be referenced elsewhere in the program by its name, like a variable.

The statements defining an arithmetic subprogram must be written at the beginning of the program, before the executable statements.

An example of an arithmetic subprogram is that set up to evaluate the square of the distance of a point to the origin of coordinates, that is, the sum of the squares of its coordinates:

$$\text{DIS}(X, Y, Z) = X ** 2 + Y ** 2 + Z ** 2$$

This arithmetic statement will be executed each time the name of the arithmetic subprogram,  $\text{DIS}(X, Y, Z)$ , is encountered in the main program; the arguments  $X$ ,  $Y$  and  $Z$  may be replaced with any other, for example,  $(A, B, C)$ . In this way, an arithmetic subprogram makes it unnecessary to write the same sequence of operations several times.

**FUNCTION subprograms.** A function subprogram is a complete computational procedure in its own right; it even has an `END` statement of its own to indicate the physical end of the procedure.

A `FUNCTION` subprogram is fully independent of, and separated from, the main program. This subprogram presents a list of statements executable each time the main program mentions the name assigned to the `FUNCTION` subprogram.

A `FUNCTION` subprogram differs from the main program in the first statement. This statement is composed of a keyword, `FUNCTION`, the subprogram name, and a list of arguments enclosed in parentheses. The name of a `FUNCTION` subprogram obeys all the rules applicable to any `FORTRAN` names.

The first statement of a `FUNCTION` subprogram may be a line of the form:

`FUNCTION IFACT (IARG)`

provided the subprogram name is `IFACT`.

A `FUNCTION` subprogram differs from the usual programs also in that its last executable statement returns control to the main program. This statement is a single keyword, `RETURN`. Ordinarily, it is placed immediately before the `END` statement of the subprogram.

Each `FUNCTION` subprogram has one more mandatory statement. This statement assigns the result of the execution to the variable whose name is the same as that of the subprogram (less parentheses and list of arguments). This statement may be placed anywhere, but most often it occurs immediately before the `RETURN` statement.

As with library and arithmetic subprograms, the main program references a `FUNCTION` subprogram by its name. Also as with library and arithmetic subprograms, the name of a `FUNCTION` subprogram defines the form in which to present the result.

**SUBROUTINE subprograms.** These subprograms are not unlike `FUNCTION` subprograms in many respects. For one thing, both types must contain `RETURN` and `END` statements; both are fully

independent of, and separated from, the main program. On the other hand, while a FUNCTION subprogram returns a single value associated with the function name, a SUBROUTINE subprogram may return none, one or more than one value. A FUNCTION subprogram is called in by the main program by reference to its name. With a SUBROUTINE subprogram this is done by means of a special statement, CALL. This statement is composed of a keyword, CALL, the subprogram name, and a list of names of its arguments, enclosed in parentheses.

SUBROUTINE subprograms may contain CALL statements to reference other SUBROUTINE subprograms and also any other statements referring to FUNCTION, arithmetic or library subprograms. A SUBROUTINE subprogram may not call itself either directly, or indirectly, through a chain of other subprograms. This limitation also applies to FUNCTION subprograms.

### 8.3.7. Examples of FORTRAN Programs

#### Example 1

```

C      PROGRAM TO EVALUATE THE ROOTS OF A QUAD-
        RATIC EQUATION
      1  FORMAT (3F10.4)
        READ 1, A, B, C
    10  FORMAT ('COEFFICIENTS', 2X, 'A=', F6.4, 4X,
        * 'B=', F6.4, 4X, 'C=', F6.4)
        WRITE 10, A, B, C
C      EVALUATE DISCRIMINANT
        DISCR=B * B - 4. * A * C
        IF(DISCR)8, 2, 3
C      TWO EQUAL ROOTS
      2  X= -B/(2. * A)
      4  FORMAT ('TWO EQUAL ROOTS', 2X, 'X=', F6.4)
        WRITE 4, X
        GO TO 7
C      TWO DISTINCT REAL ROOTS
      3  R=SQRT(DISCR)
        X1=(-B + R)/(2. * A)
        X2=(-B - R)/(2. * A)
      5  FORMAT ('TWO REAL ROOTS', 2X,
        * 'X1=', F7.4, 4X, 'X2=', F7.4)
        WRITE 5, X1, X2
        GO TO 7
C      TWO COMPLEX ROOTS
      8  X1= -B/(2. * A)
        Y1=SQRT(-DISCR)/(2. * A)
        Y2= -Y1
      6  FORMAT ('TWO COMPLEX ROOTS', 2X, 'R1=1',
        * F8.4, 2X, 'IM1=+', F8.4, ',', 2X, 'R2=',

```

```
      * F8.4, 2X, 'IM2=', F8.4)
      WRITE 6, X1, Y1, X2, Y2
7  STOP
  END
  COEFFICIENTS: A=4.0000 B=6.0000 C=2.0000
  TWO REAL ROOTS: X1= -0.50000 X2= -1.0000
```

**Example 2**

```
C  PROGRAM TO FIND A MAXIMUM
    DIMENSION X(50)
    5  FORMAT ('MAXIMUM ELEMENT
      * X(', 12, ') =', F7.4)
    1  FORMAT (10F7.4)
      READ 1, X
      XMAX=X(1)
      K=1
      DO 2 I=2,50
        IF(XMAX - X(I)) 3, 2, 2
    3  XMAX=X(I)
      K=1
    2  CONTINUE
      WRITE 5, K, XMAX
      STOP
      END
      MAXIMUM ELEMENT X(17)=5.4783
```



# Chapter IX

## Special-purpose Computers

### 9.1. Control Computers

**9.1.1. General.** Apart from handling a great variety of mathematical problems, present-day high-speed electronic computers have proved their worth in automatic control of processes, plants and systems. Advances in computer engineering have given big impetus to cybernetics, the science of control and communication in the animal and the machine (as N. Wiener put it), and notably engineering cybernetics concerned with direct applications of cybernetics to engineering systems. To-day, automatic control is being applied in a multitude of industries, branches of sciences and technology from automated systems in metallurgy, chemistry, mechanical engineering and power generation to highly sophisticated aerospace systems.

A control computer is an arrangement of devices which operate on input information about the behaviour of the associated controlled process, plant or system so as to control or guide it under a prescribed algorithm. A control computer and the associated controlled plant make up a closed-loop or feedback system in which all the necessary input process variables are fed to the computer for computation, evaluation and generation of an optimum solution. The solution is then converted to control signals which go to appropriate controllers and/or actuators to change the behaviour of the plant so as to achieve an optimum response. The changed process variables are again routed into the computer, and the chain of events is repeated again and again.

**9.1.2. Basic types and applications.** Control computers may be general-purpose, intended to control a group of processes or plants that can be described by a similar or identical mathematical formalism, and special-purpose, having a simpler structure and used to control a single specific process.

Depending on the control logic involved and the accuracy required, a control computer may use analog or digital computing elements, or both. As a rule, analog computing elements are used where control is concerned with less complex processes and the accuracy requirements are not particularly stringent. Among their advantages are freedom from analog to digital conversion for input and the reverse conversion for output. Digital computing elements are a natural choice in applications involving complex processes and plants. Among the advantages of a digital control

computer are large-capacity memory, broad capabilities of logic, flexibility, and high accuracy in data processing.

As an example, Fig. 9.1 shows a block diagram of an automatically controlled industrial system where control is effected by a control computer. The controlled plant is subjected to a number of fixed uncontrolled (disturbance and load) variables,  $p_q$  ( $q = 1, \dots, k$ ). Suitable transducers,  $T_i$ , sense the status of the plant by measuring changes in some of its variables,  $x_i$  ( $i = 1, \dots, n$ ). These output or state variables are routed via a scanner,  $Sc1$ , to an analog-to-digital converter,  $ADC$ , in a pre-arranged sequence. Owing to the scanner, a single multi-channel  $ADC$  can be used, thereby giving a sizeable economy in the equipment needed. In the  $ADC$ , an analog quantity,  $AQ$ , is coded into

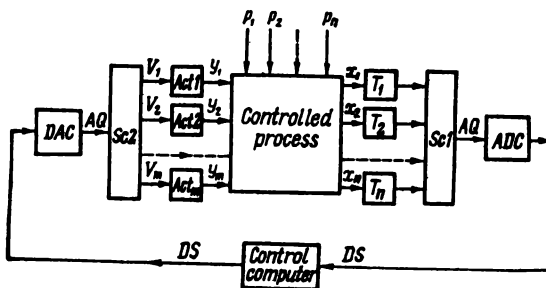


Fig. 9.1

a digital signal,  $DS$ , which is read into the computer memory. Once it has stored all the relevant input data, the computer sets out on a computational procedure. The outcome of this computation is a set of digital signals. Before they can be used by appropriate controllers, they must be (and are) converted back to analog form by a digital-to-analog converter,  $DAC$ , and routed as voltages,  $V_j$  ( $j = 1, \dots, m$ ), via another scanner,  $Sc2$ , to the respective actuators,  $Act_j$ , which change the manipulated variables,  $y_j$ , so as to bring about a desired change in the status of the process or plant.

A control algorithm for a closed-loop automatic control system usually has provisions for optimization of control on the basis of the plant behaviour and changes in the environmental conditions and in accordance with some preselected criteria. Such systems are sometimes called self-organizing or adaptive. In contrast, there are systems which operate in a rigid manner to a predetermined program executed irrespective of the changes that may actually take place in the process or the environments. Finally, there are automatic control systems which anticipate changes in

the process variables due to a pre-selected set of environmental conditions. In the last case, control computers are often of the analog type.

## 9.2. Digital Differential Analyzers

**9.2.1. General.** These machines solve ordinary differential equations by making use of a numerical approximation to integration, for which reason they are often called digital integrators.

Digital differential analyzers (*DDA*) can be used as such and also as part of automatic control systems. In the latter case, they operate under a preset computational and control program, automatically switch programs and the classes of problems to be accommodated in accordance with the results obtained or changes in some variables.

Present-day digital differential analyzers have speeds sufficient for real-time simulation of complex dynamic systems operating at very high speeds, and also for use in conjunction with actual processes and plants. The capabilities of digital differential analyzers, including their accuracy and speed, can materially be improved through the use of multi- and full-digit instead of single-digit incrementation, floating-point arithmetic, and integration by successive approximation. Also, a *DDA* can readily be ganged up with a conventional digital or analog computer.

The basic computing element of a *DDA* is an operational digital integrator. Apart from such integrators, a *DDA* also includes converters, memory, control, logic elements, etc.

**9.2.2. Digital operational integrators.** In the general case, a digital operational integrator establishes a relation of the form

$$z = \int_a^b y \, dx = \int_a^b f(x) \, dx \quad (9.1)$$

where all variables are numbers, that is, they are in discrete form, and the independent variable may be chosen arbitrarily. Integration is carried out numerically, by adding together the increments, using rectangular or trapezoidal techniques. With rectangular integration, the integral of the specified function can be approximated by a sum of the areas of elementary rectangles, each representing a specific increment,  $\Delta x$  (Fig. 9 2):

$$z = \int_a^b y \, dx = \lim_{n \rightarrow \infty} \sum_{i=1}^n y_i \Delta x_i \approx \sum_{i=1}^n y_i \Delta x_i$$

where  $y_i$  is the ordinate of an  $i$ th rectangle. Assuming that  $\Delta x = h = 1$ , then

$$z = \sum_{i=1}^n y_i \quad (9.2)$$

that is, the operation of integration is reduced to adding together the ordinates  $y_i$  of the rectangles. The ordinates  $y_i$  are given by

$$y_i = y_0 + \sum_{k=1}^i \Delta y_k \quad (9.3)$$

where  $y_0$  is the initial value of the function.

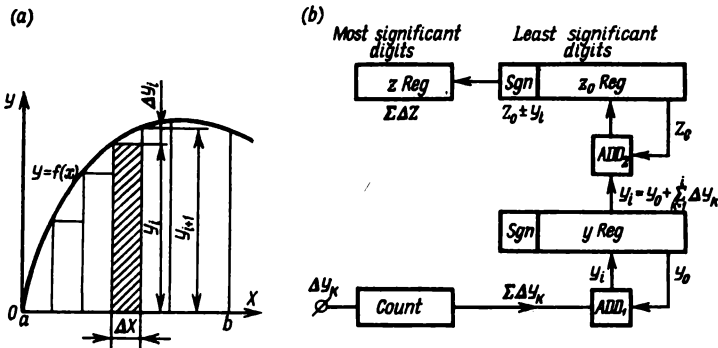


Fig. 9.2

Equations (9.2) and (9.3) are mechanized by the integrator whose circuit is shown in Fig. 9.2b. To facilitate the connection of integrators to one another and to other computing elements, the increments  $\Delta x_i$ ,  $\Delta y_i$  and  $\Delta z$  are represented as unit pulses. The increments  $\Delta y_k$  (0 or 1) are fed to a reversible counter, count, which keeps the running total of unit pulses,  $\Sigma \Delta y_k$ , that is, the increment in the integrand. Then an adder,  $ADD1$ , adds  $\Sigma \Delta y_k$  to the initial value  $y_0$  to generate a new ordinate  $y_i$  for each integration step in accordance with Eq. (9.3). The ordinates  $y_i$  are accumulated by a  $y$  register. A  $\Delta x$  pulse causes another adder,  $ADD2$ , to add or subtract  $y_i$  to or from  $z_0$  previously accumulated in the output  $z_0$  register. As a result, the  $z_0$  register stores a number representing  $\Sigma y_i$ , that is the value of the integral  $z$ . The  $y$  and  $z_0$  registers have each  $n$  bits sufficient to represent the number  $2^n$ . Adding together two  $n$ -bit numbers  $y_i$  and  $z_0$  may cause the  $z_0$  register to overflow, in which case an overflow pulse,  $\Delta z$ , will appear at its output (a carry to the next most significant digit), which is recognized as a unit pulse increment in the integral  $z$ . The overflow pulses,  $\Delta z_i$ , are accumulated by the  $z$  register. Thus,

the  $z_0$  and  $z$  registers may be looked upon as a single  $2n$ -bit register in which one part (the  $z_0$  register) stores the least significant bits, and the other part (the  $z$  register) stores the most significant bits of the integral  $z$ . The increment in the integral is given by

$$\Delta z = ky \Delta x$$

where

$$k = 1/2^n = \text{constant}$$

If  $y = 1$  and  $\Delta x = 1$ , then  $2^n$  integration steps will yield one overflow pulse, that is

$$\Delta z = (y \Delta x / 2^n) 2^n = 1$$

Practical digital integrators usually have no  $z$  registers and the overflow pulses,  $\Delta z$ , are directly taken from the register output to another device. The type of operation to be carried out by *ADD2* depends on whether a given  $\Delta x_i$  pulse is a 1 or a 0. If it is a 1, the adder will carry out the operation of addition,  $z_0 + y_i$ ; in the latter case the operation of subtraction,  $z_0 - y_i$ .

As an example, consider a case where the initial value of the integral is  $z_0 = 0.00 \dots 00$ , and the integrand  $y$ 's a one in the least significant digit position, that is,

$$y_0 = \boxed{1} 00 \dots 01 = \text{constant}$$

The leftmost digit position is the sign bit. If it contains a 1, the number is positive; if it contains a 0, the number is negative. Each integration step, that is, each addition of  $z_0$  and  $y_i$  will increment the contents of the  $z_0$  register by one. As a result, a positive or a negative overflow will always occur. It is assumed that the presence of an overflow pulse corresponds to the "+" sign, and its absence, to the "-" sign.

The first integration step may be written as:

$$\begin{array}{r} z_0 \boxed{1} 00 \dots 00 \\ + y_0 \boxed{1} 00 \dots 01 \\ \hline 1 \boxed{0} 00 \dots 01 \end{array}$$

The 1 beyond the sign bit represents an overflow pulse.

The second integration step may be presented as:

$$\begin{array}{r} z_0 \boxed{0} 00 \dots 01 \\ + y_0 \boxed{1} 00 \dots 01 \\ \hline 0 \boxed{1} 00 \dots 10 \end{array}$$

As is seen, the overflow bits at the integrator output are an alternating sequence of ones and zeros. This is another way of saying that unit increments come in an alternating sequence of  $+1$ 's and  $-1$ 's:

$$\Delta z = +1, -1, \dots$$

The net result will be zero. After  $2^n$  additions, the alternating sequence of unit increments will cease, and a unit overflow pulse,  $\Delta z$ , will appear at the output.

A digital integrator may be based on serial or parallel adders. In the former case, the amount of equipment needed is smaller and the circuit is therefore simpler, but the speed of operation is reduced. Serial registers may be magnetic-drum or other cyclic memory devices.

### 9.2.3. Serial and parallel digital differential analyzers

#### *Serial DDA*

In a serial digital differential analyzer, a single set of computing elements is used to process several integrators in turn. The numbers associated with each integrator (operands, partial and final results) and also the computational procedures are stored in a memory unit. Because of the inherent reliability and compactness, the storage device most commonly used is a magnetic drum with parallel tracks on its surface. Referring to a simplified block diagram of a serial *DDA* in Fig. 9.3, the  $z$  and  $y$  tracks are reserved to contain serially the  $z_0$  and  $y$  registers for each of the digital integrators, while the  $\Delta z$  track is reserved for the  $\Delta z$  overflow pulses of all the integrators. The  $k_x$  and  $k_y$  tracks store addresses for integrator interconnections. A "1" written into a digit position on the  $k_x$  or  $k_y$  track will, on moving past the read head, *RH*, cause the appropriate AND gate, 1 or 2, to open and gate out a  $\Delta z$  signal from the  $\Delta z$  track to the  $\Delta x$  or  $\Delta y$  input of one of the integrators; the  $\Delta z$  values are transferred. Synchronism in operation of the *DDA* units is maintained by clock pulses, *CP*, which are applied, beginning at the instant when the "start" pulse (*SP*) written on a separate track is read. Each of the  $z$ ,  $y$  and  $\Delta z$  tracks is divided into segments, each acting as an integrator register (Fig. 9.3b). For example, if segments 1, 2, 3 and 4 store operands, these are read by the read head and transferred for computations, the results of which are simultaneously written by the write head into segments 1', 2', 3' and 4'. Following each integration step, which corresponds to a drum rotation of  $180^\circ$ , the previous record is erased to be replaced with a new one or restored again. Connection of segments to adders produces digital

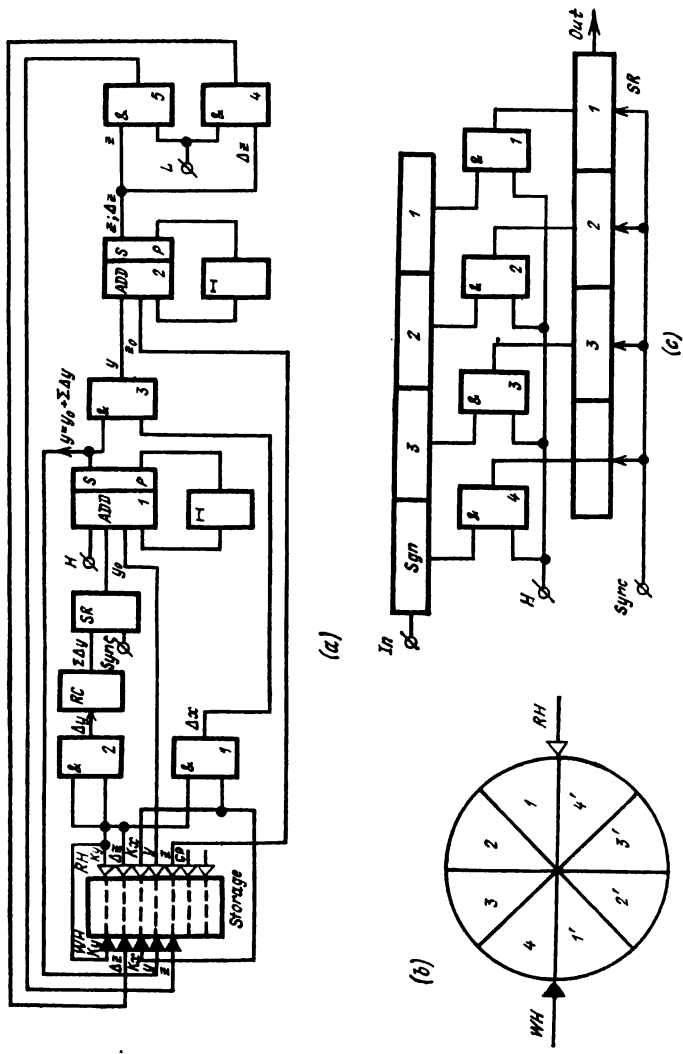


Fig. 9.3

integrators. During each step of integration adders 1 and 2 mechanize the following equations:

$$y_i = y_0 + \sum_{k=1}^i \Delta y_k$$

$$z = z_0 + y_i$$

where  $y_0$ ,  $z_0$ ,  $y$  and  $z$  are the initial and final values of the integrand and integral, respectively, for each step of integration.

The sum  $\Sigma \Delta y$  must be formed before integration; for this purpose the  $\Delta z$  read and write heads are shifted one sector relative to the  $y$  and  $z$  read heads. The value of  $\Sigma \Delta y$  is formed by the reversible counter, *RC*, which accumulates the  $\Delta z$  pulses arriving by way of the open AND gate, *AND2*. Then  $\Sigma \Delta y$  is applied to a shift register, *ShR* (Fig. 9.3c) which in response to clock pulses, shifts the number and transfers it sequentially as a train of pulses to *ADD1*, starting with the least significant digits. At the same time, *ADD1* accepts the value of  $y_0$  from the  $y$  track likewise with the least significant bits first. The new value,  $y = y_0 + \Sigma \Delta y$ , is fed via the AND gate (*AND3*), which is open for  $\Delta x = 1$  and closed for  $\Delta x = 0$ , to *ADD2* and the write head of the  $y$  track for the write operation. The adder, *ADD2*, forms a new value of the integral,  $z = z_0 + y$ , which is routed via the open AND gate, *AND5*, remaining open as long as clock pulses 1 through  $n$  arrive, and closing on the  $(n + 1)$ st clock pulse,  $L$ , which represents the last (most significant) digit of the number  $z$ . However, the  $L$  pulse causes the AND gate, *AND4*, to open and gate out a  $\Delta z$  overflow pulse which is then written on the  $\Delta z$  track. All subsequent steps of integration are identical to that described above.  $\Delta z$ ,  $\Delta y$ ,  $\Delta x$  and  $y$  may be positive, as was tacitly assumed in the above description, or negative. If both positive and negative quantities have to be accommodated, a more elaborate circuit must be used.

### Parallel DDA

In a parallel digital differential analyzer, all integrators are processed at the same time, that is, in parallel. In fact, this is a parallel combination of several digital integrators, each with adders and other computing elements of its own. Obviously, there is no need for a common memory. While for a serial DDA the time needed for a solution is

$$t_s = nt_i$$

where  $t_i$  is the time taken by one integrator to perform integration and  $n$  is the number of integrators used, in a parallel DDA,  $t_s$  is decided by the time required for a single integration and is independent of the number of integrators.



Integrators are usually interconnected via a special panel much as is done for serial digital differential analyzers. Because of this "rigid" arrangement, the logic capabilities of a parallel *DDA* are limited. On the other hand, parallel *DDAs* offer a high speed of operation. As an example, the Soviet-made METEOR-3 parallel *DDA* using 100 digital integrators and occupying a space of as little as 2 cu.m can do 4 million basic operations per second. The more recent models of parallel *DDAs* can do 10 to 20 million basic operations (such as additions) per second, which is in no way a final limit. Such speeds are beyond the reach of digital computers.

The latest advances in microelectronics hold out special promise for a further decrease in the size and a further improvement in the performance of digital differential analyzers.

It may be added that the advantages of serial and parallel digital differential analyzers can be combined in serial/parallel machines.

### 9.3. Analog-Digital Computers

Of late, there has been a growing trend to combine the advantages of analog computers (high speed, simplicity of problem preparation and set-up) and digital computers (high accuracy and broad logical capabilities) in combined schemes.

A combined analog-digital system may be synthesized in a variety of ways. Most often, such a system has an analog section and a digital section, each using only analog and only digital data representation, respectively. The digital section may essentially be a digital computer, a digital differential analyzer, or a set of digital computing elements. The analog section may use an analog computer or a set of analog operational elements. Data transfer between the two sections in the course of a computation is effected by converters. Synchronism in operation of the two sections may be maintained either by a separate control unit or by the control unit of the digital computer.

As a rule, the analog section of a combined system simulates short-duration processes where the requirements for accuracy are modest, and the digital section provides statistical reduction of slowly varying processes with a higher accuracy and with provision of logical operations.

Suppose we are to synthesize a combined system to solve ordinary differential equations of the form

$$\frac{dx_i}{dt} = f_i(x_1, x_2, \dots, x_n, t), \quad i = 1, \dots, n \quad (9.4)$$

The logical design to be finally chosen will depend on how the various functions are to be shared between the analog and digital sections. Most often, the digital section will be entrusted with

evaluating the integral of the precise part of Eqs. (9.4), with terms materially affecting the overall precision of the solution. Then, the analog section will take the integral of the coarse part of the equations, where high precision is not essential.

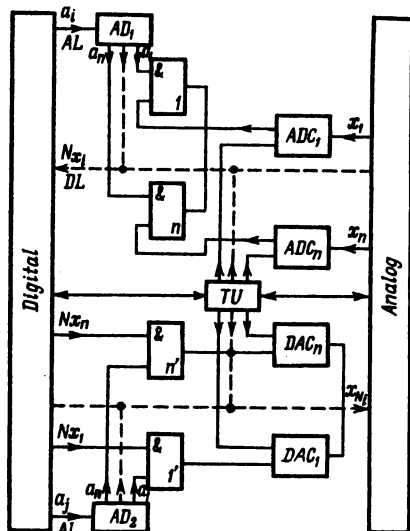


Fig. 9.4

analog conversion (DAC) channels route the analog values  $x_{n_i}$  corresponding to the numbers  $N_{x_i}$  to the analog section. All units of the combined system are kept in synchronism by a timing unit,  $TU$ .

#### 9.4. Prospects in the Development and Use of Electronic Computers

Electronic computers are continually expanding into all fields of man's activity, and especially so in science, engineering, automatic control, and data processing for accounting, planning and economic purposes, automatic information retrieval and operations research.

The major objectives in the advanced digital computer designs are greater capacity, improved reliability, reduced size and power requirements, and wider capabilities. The capacity of a digital computer is above all decided by the speed of its components and the manner in which problems are prepared for computation and the computations are carried out.

Existing computer components based on electron valves, semiconductors and magnetic (mostly ferrite) cores can give operating

rates of the order of a few megahertz to a few tens or hundreds of megahertz. Any further improvements in speed, reliability, size and power requirements necessitate the use of computer components capable of operating in the *SHF*, *EHF* and higher frequency ranges, which inevitably involves a number of difficulties. At present, the speed of digital computer elements is being improved through the use of r.f. semiconductor devices (transistors, tunnel diodes), thin and extremely thin films (fabricated from ferromagnetic, semiconductor, conducting and dielectric materials), cryogenic devices, that is devices utilizing the phenomenon of superconductivity observed at temperatures close to the absolute zero (cryosars, cryotrons, persistors with switching times of the order of  $10^{-8}$  or  $10^{-9}$  s), microwave devices (parametrons, travelling-wave tubes), masers and lasers. Some of these components have already found practical application, and some are under investigation for use in computation. An increase in computer speed can also be secured through the use of better input/output equipment, new methods for data presentation and processing. Among other things, a good deal of hope is placed in the development of devices that would read data into a computer directly from a text or by voice.

Appreciable effort is being put into the logic design of digital computers with a view to expanding computer capabilities for adaptation to changing input data.

Material improvements in computer capacity can be obtained by setting up integrated digital computer systems with an overlap of computational and other operations, with the result that several million operations can be performed every second. Unfortunately, the accompanying increase in the amount of equipment and decrease in reliability may more than offset the gain in speed. Reliability can be improved through redundancy and the use of highly stable and reliable components.

A sizeable contribution to better computer performance has been coming from microminiaturization, especially micromodules with a packaging density of 10 to 20 elements per cubic centimetre, thin-film microcircuits with a packaging density of 100 to 200 parts per cubic centimetre, and solid-state integrated (molecular) circuits with a packaging density of 1000 and more elements per cubic centimetre.

The increase in the amount of data processed calls for proportionately larger memories, especially for data processing and information retrieval systems, and this demand is being satisfied. Mention should also be made of the effort in the field of automatic computer design and manufacture, standardization of computer elements and units, and the wider use of automatic programming.

New and better types of analog computers are being developed for the investigation and design of systems by simulation techniques and also for automatic industrial control. Sizeable improvements have been achieved in size, economy, reliability and stability of analog computers through the use of miniature and highly stable semiconductor, magnetic and other devices. Advanced types of operational amplifiers and function generators coupled with more efficient methods of error detection and compensation have markedly improved the accuracy of analog computers.

The versatility of analog computers has been greatly expanded with the incorporation of digital elements (memory, logic circuits, I/O devices, etc.). An important problem awaiting its solution is further cutback in the time an average analog computer needs to solve a problem, through the use of automatic input/output procedures, solution monitoring and control, problem preparation and set-up, and solution display.

An attractive prospect is seen in further development of special-purpose computers possessing high accuracy, high speed, high reliability and small size, particularly in conjunction with better methods and hardware for automatic process control, and also high-performance digital differential analyzers, hybrid and combined systems.

## Bibliography

1. Anisimov V. B., Chetverikov V. N. *Osnovy teorii i proektirovaniya ETsVM (Fundamentals of the Theory and Design of Electronic Digital Computers)*, Moscow, "Higher School", 1970.
2. Bruevich N. G., Dostupov B. G. *Osnovy teorii schetno-reshaiushchikh ustroystv (A Basic Theory of Computing Devices)*, Moscow, "Sovetskoe radio", 1964.
3. Vitenberg I. M. *Programmirovaniye analogovykh vychislitelnykh mashin (Programming for Analog Computers)*, Moscow, "Mashinostroenie", 1972.
4. *Vychislitelnye mashiny nepreryvnogo deistviya (Analog Computers)*. Ed. by V. B. Smolova and A. N. Lebedev, Moscow, "Higher School", 1964.
5. *Vychislitel'naya tekhnika (Computers and Computation)*. Handbook, vol. I and II. Translated from English. Ed. by A. V. Shileiko. Moscow — Leningrad, "Energia", 1964.
6. Gitis E. I. *Preobrazovateli informatsii dlya elektronnykh tsifrovyykh vychislitelnykh ustroystv (Transducers and Converters for Electronic Digital Computers)*. Moscow, Gosenergoizdat, 1961.
7. Glushkov V. M. *Sintez tsifrovyykh avtomatov (Logical Design of Digital Circuits)*. Fizmatgiz, 1962.
8. Gnedenko B. V., Koroluk V. S., Yushchenko E. L. *Elementy programmirovaniya (Principles of Programming)*. Fizmatgiz, 1961.
9. Dobrogurski S. O., Kazakov V. A., Titov V. K. *Schetnoreshayushchie ustroystva (Computers)*. Moscow, Oborongiz, 1959.
10. Kitov A. I., Krinitsky K. A. *Elektronnye tsifrovye mashiny i programmirovaniye (Electric Digital Computers and Programming)*. Fizmatgiz, 1961.
11. Kogan B. Ya. *Elektronnye modeliruyushchie ustroystva i ikh primeneniye dlya issledovaniya sistem avtomaticheskogo regulirovaniya (Electronic Analogs and Their Use in the Study of Automatic Control Systems)*. Fizmatgiz, 1959.
12. Kraizmer L. P. *Bystrodeistvuyushchie ferromagnitnye zapominayushchie ustroystva (High-Speed Ferromagnetic Memory Elements)*. Moscow, "Energia", 1964.
13. Mayorov F. V. *Elektronnye tsifrovye integriruyushchie mashiny (Electronic Digital Integrators)*. Moscow, Mashgiz, 1962.
14. Pervin Yu. A. *Osnovy FORTRANA (An Outline of FORTRAN)*, Moscow, "Nauka", 1972.
15. Portnoi G. P., Vavilov E. N. *Sintez skhem elektronnykh tsifrovyykh mashin (Logical Design of Electronic Digital Computers)*. Moscow, "Sovetskoe radio", 1963.
16. Prokhorov V. I., Pogorelko I. A., Yakovlev V. A. *Osnovy programmirovaniya dlya elektronnykh vychislitelnykh mashin (Fundamentals of Programming for Electronic Computers)*, Moscow, "Higher School", 1967.
17. Smolov V. B. *Analogovye vychislitelnye mashiny (Analog Computers)*. Moscow, "Higher School", 1972.
18. Suchilin A. M. *Osnovy vychislitel'noy tekhniki (Fundamentals of Computers and Computation)*. Moscow — Leningrad, "Energia", 1964.

## A.

adders, 194  
—, multi-bit, 197  
—, —, parallel, 197  
—, —, serial, 197  
—, using accumulators, 195  
—, using separate sum register, 197  
address, 214  
analog computers, committed, 99  
—, solving of linear algebraic equations, 123  
—, solving of nonlinear equations, 115  
—, uncommitted, 99, 101  
analogs, conductive-liquid, 25  
—, conductive-sheet, 25  
—, continuous-field, 24  
—, resistance-network, 26, 30  
analog-to-digital conversion, comparison method, 231  
—, feedback method, 234  
—, frequency method, 233  
—, incremental method, 231  
—, pulse-counting method, 233  
—, total-value method, 231  
analog-to-digital converters, 227, 231  
analogy, direct, 13  
arithmetic unit, 127, 201  
—, logical design for addition and subtraction, 201  
—, logical design for division, 203  
—, logical design for multiplication, 202

## B.

base of number system, 131  
binary-coded decimal numbers, 135  
Boolean algebra, 142  
Boolean algebra, basic laws, 146  
Boolean expressions, 146  
Boolean functions, 144, 177  
—, basic definitions, 144  
—, functional completeness, 149  
—, minimization of, 150  
—, —, by Quine method, 151  
—, —, by Veitch diagrams, 153

## C.

card punch, parallel, 237  
—, serial, 238  
cards, punched, 236  
—, preparation of, 240  
cathode-ray-tube function generators, 82  
character display devices, 243  
Charactron tube, 243  
chopper amplifier, 48  
chopper stabilization, 48  
chopper-stabilized d. c. amplifier, 48  
code, Gray, 235  
—, cycle permuted, 235  
—, cyclic, 235  
—, straight binary, 235  
code wheel, 234  
coding, 252  
coding disc, 234  
coefficient-setting potentiometers, 95  
crossed-field electron-beam multipliers, 92  
combinational circuits, 142  
combinational networks, 178  
—, multi-output, 183  
—, single-output, 179  
—, synthesis of, 179  
Compositron tube, 243  
computational loops, 264  
computers, analog, 9, 11-126  
—, analog-digital, 295  
—, applications, 9  
—, classification, 9  
—, digital, 9, 127-251  
—, hybrid, 10  
—, special-purpose, 287  
computing elements, 34 96  
contracted disjunctive normal form, 151  
control unit, 128, 245  
converters, analog-to-digital, 227, 231  
—, digital-to-analog, 228  
—, shaft-position 234  
counters, 190  
—, binary, up-, 191  
—, reversible, 192  
current summation, 55  
cycle, loop, 265

## D.

data reading, electric-contact, 239  
 —, electromechanical, 239  
 —, from punched cards, 239  
 —, from punched tape, 239  
 —, photoelectric, 239  
 debugging, 252  
 decoders, 198  
 —, matrix, 198  
 —, pyramidal, 199  
 differential analyzers, analog, 97  
 —, synthesis of, 97  
 —, block diagram of, 97  
 —, digital, 289  
 —, —, parallel, 294  
 —, —, serial, 292  
 —, main units of, 102  
 —, problem preparation for, 106  
 —, problem solving on, 113  
 differentiation by tachometer generators, 59  
 differentiators, 58  
 —, electrical, 61  
 —, electronic, 61  
 digit time, 142  
 digital circuits, 142  
 digital computers, asynchronous, 246  
 —, automatic program solving on 129  
 —, basic parameters of, 127  
 —, elements of, 156  
 —, logical assemblies of, 186  
 —, logical design of, 177  
 —, mathematical basis of, 127  
 —, problem preparation for, 260  
 —, single-address, 260  
 —, structure of, 127  
 —, synchronous, 246  
 —, three-address, 260  
 digital operational integrators, 289  
 digital-to-analog converters, 228  
 diode function generators, 70  
 —, special-purpose, 75  
 —, Zener diode, 78  
 diode-transistor logic, 163  
 disjunctive normal form, contracted, 151  
 —, dead-end, 151  
 —, expanded, 148  
 —, minimal, 151  
 drift, 45  
 —, in transistor operational amplifiers, 51  
 —, parametric compensation of, 47  
 DTL, 163

## E.

Eccles — Jordan circuit, 174  
 emitter follower, 162  
 equation solvers, adjuster-type, 118  
 —, algebraic, 118  
 —, servo-operated, 119  
 —, transcendental, 118  
 equation-solving, 13

## F.

ferrite core, biaxial-aperture, 209  
 —, secondary-aperture, 209  
 ferrite core-diode logic, 168  
 ferrite core-transistor logic, 171  
 flip-flop, basic, 174  
 —, dynamic, 176  
 —, set-reset, 176  
 —, set-reset-trigger, 176  
 —, trigger, 176  
 —, typical circuits, 174  
 FORTRAN, 269-86  
 —, arithmetic statements, 273  
 —, character set, 270  
 —, constants, 271  
 —, CONTINUE statement, 277  
 —, control statements, 275  
 —, DO statement, 279  
 —, END statement, 279  
 —, FORMAT statement, 280  
 —, GO TO statement, 275  
 —, IF statement, 275  
 —, input-output of alphabetic information, 282  
 —, I/O statements, 279  
 —, loops, 275  
 —, number range, 273  
 —, PAUSE statement, 278  
 —, precision, 273  
 —, READ statement, 279  
 —, STOP statement, 278  
 —, subprograms, 282  
 —, variables, 271  
 —, WRITE statement, 281  
 function approximation, 67  
 —, piecewise-linear, 67  
 function dividers, 85  
 function generation, 114  
 function generators, 66  
 —, cathode-ray-tube, 82  
 —, diode, 70  
 —, shaped-potentiometer, 80  
 —, special-purpose, 83  
 —, stepping-switch, 79  
 —, time, 79  
 function multipliers, 85

## G.

gate, AND, diode, 161  
 —, —, diode-transformer, 161  
 —, NOT, 162  
 —, —, transistor, 161  
 —, OR, diode, 161

## H.

half-current, 214  
 Hall constant, 91  
 Hall effect multipliers, 90  
 Hall generator, 90  
 hypothetical computer, 253

## I.

induction resolvers, 36  
 input devices, 239  
 input media, 236  
 input-output equipment, 227  
 input unit, 128  
 instruction word, 129  
 instructions, 129  
 —, branch, 263  
 —, loop, 264  
 integrated circuits, logical, 165  
 integration by tachogenerators, 59  
 integrators, 58  
 —, digital operational, 289  
 —, electrical, 61  
 —, electric-network, 30  
 —, electronic, 61  
 internal memory, ferrite-core, 221  
 —, control of, 221  
 internal storage, organization of, 214  
 inverter, 161  
 iteration, Gauss — Seidel method, 125

## L.

language, algorithmic, 252  
 linear algebraic equations, solving, by  
 characteristic equations, 124  
 —, —, by iteration methods, 125  
 —, —, by minimization, 126  
 —, —, by root adjustment, 123  
 logic, diode-transistor, 163  
 —, ferrite core-diode, 163  
 —, ferrite core-transistor, 171  
 logic assemblies, 186  
 logical design, 177  
 logical elements, 156  
 —, bistable, magnetic-core, 165, 181  
 —, semiconductor, 157

logical networks, 177  
 loop iteration, 265  
 loop cycle, 265

## M.

machine cycle time, 246  
 magnetic head, air-floated, 219  
 Mealy automata, 143  
 memory, coincident-current, 216  
 —, delay-line, 213  
 —, external, control of, 224  
 —, fixed, 223  
 —, read-only, 223  
 —, word-organized, 216  
 memory cell, 214  
 memory unit, 128  
 microinstructions, 251  
 microprogramming, 251  
 modelling, 11  
 —, mathematical, 12-5  
 —, physical, 11-2  
 —, physical fields, 23  
 models, electric, for physical systems,  
 18  
 —, mathematical, 20  
 —, —, synthesis of, 20  
 —, physical, 18  
 —, —, synthesis of, 18  
 Moore automata, 143  
 multi-aperture ferrite plates, 208  
 multipliers, AM, 89  
 —, electron-beam, cross-field, 92  
 —, FM, 89  
 —, Hall effect, 90  
 —, quarter-square, 93  
 —, time-division, 88

## N.

negative numbers, 1's complement  
 form, 134  
 —, 2's complement form, 133  
 —, sign-and-magnitude representation,  
 133  
 numbers, fixed-point, 133  
 —, —, addition, 136  
 —, —, division, 139  
 —, —, subtraction, 136  
 —, —, taking product of, 138  
 numbers, floating-point, 133  
 —, —, addition, 137  
 —, —, division, 139  
 —, —, subtraction, 137  
 —, —, taking product of, 138  
 number range, digital, 261  
 number system, binary, 131



number, hexadecimal, 131  
 —, octal, 131  
 number systems, 131  
 —, positional, 131  
 —, conversion between, 132

## O.

operation code, 129  
 operational amplifier, 42-54  
 —, as differentiator, 44, 46  
 —, as integrator, 44, 46  
 —, as scaler, 44, 46  
 —, as summer, 44, 46  
 —, as summing integrator, 44, 46  
 —, chopper-stabilized, 48  
 —, differentiating, 63  
 —, integrating, 63  
 —, transfer function of, 47  
 —, transistor, 50  
 —, valve, 45  
 output devices, 241  
 output unit, 128

## P.

physical fields, modelling of, 23  
 polynomial evaluators, 118, 120-23  
 potentiometers, 35  
 —, coefficient-setting, 95  
 —, shaped-card, 80  
 —, tapered, 80  
 —, tapped, 68  
 potentiometer dividers, 86  
 potentiometer multipliers, 86  
 printers, 241  
 —, electrochemical, 244  
 —, electrographic, 244  
 —, electrostatic, 244  
 —, electrothermal, 244  
 —, glow-discharge-tube, 243  
 —, magnetographic, 244  
 —, matrix, 242  
 —, mechanical, 241  
 —, nonmechanical, 243  
 —, on-the-fly, 241  
 —, parallel, 241  
 —, serial, 241  
 problem, boundary-value, Dirichlet, 23  
 —, —, first, 23  
 —, —, Neumann, 23  
 —, —, second, 23  
 problem preparation, 260  
 problem solving, automatic, 129  
 problems, external-field, 24  
 —, internal-field, 23  
 programming, direct, 256

—, principles of, 252  
 —, sequential, 258  
 programs, branching, 263  
 pulse-counting circuits, 233

## Q.

quantization, amplitude, 227  
 —, time, 227  
 quantizing, 227  
 quarter-square multipliers, 93

## R.

radix of a number system, 131  
 rate generator, 40  
 read amplifier, 220  
 reader, photoelectric, 240  
 registers, 186  
 —, shift(ing), 186  
 —, —, using level elements, 189  
 —, —, using pulse-level elements, 188  
 root adjustment in equation solving,  
 123  
 round-off, 142

## S.

sampling, 227  
 scale equations, 34  
 scale factors, 34  
 —, amplitude, 34  
 —, time, 35  
 scalars, 193  
 scaling circuits, 193  
 secondary-aperture ferrite core, 209  
 sequential circuits, 142  
 sequential networks, 178  
 —, logical design of, 184  
 —, —, by canonical-form method, 184  
 shaft digitizers, 234  
 similarity variables, determination of,  
 15  
 —, —, by dimensional analysis, 16  
 —, —, by equation analysis, 15  
 simulation, backlash, 78  
 —, by direct analogies, 13  
 —, by equations, 13  
 —, by indirect analogies, 14  
 —, Coulomb friction, 76  
 —, dead-zone, 77  
 —, direct  
 —, hysteresis, 76  
 —, on-off response, 76  
 —, relay response, 76  
 —, resistance-network, 29  
 storage, 204  
 —, coincident-current, 214  
 —, control of, 220

- , matrix, 214
- , three-coordinate, 214
- , word-organized, 216
- storage elements, 171, 207
- , magnetic-core, 207
- storage location, 214
- storage media, 207
- , magnetic, 211
- , transfer of data, 237
- subroutines, standard, 269
- summation, current, 55
- , voltage, 57
- summers, 54-8
- summing networks, 54-8
- , bridge, 54
- , potentiometer, 55
- , rheostat, servo-operated, 54
- switching circuits, 142
- , asynchronous, 143
- , synchronous, 143
- , with memory, 142
- , without memory, 142
- switching functions, 177

- synchro resolvers, 36
- , linear, 36
- , scaling, 36
- , sine-cosine, 36-8

## T.

- tachometer generators, 40
- , a. c., 41
- , d. c., 40
- , differentiation by, 59
- , induction, 41
- , integration by, 59
- tank, electrolytic, 25
- tape, magnetic, 237
- , punched, 236
- , —, preparation of, 240
- truth tables, 144, 177
- Typotron tube, 243

## V.

- verifiers, 239

## TO THE READER

Mir Publishers welcome your comments on the content, translation, and design of the book.

We would also be pleased to receive any suggestions you care to make about our future publications.

Our address is:

USSR, 129820,  
Moscow, I-110, GSP,  
Pervy Rizhsky Pereulok, 2,  
Mir Publishers







**PROFESSOR VASHKEVICH N.P.,**  
Doctor of Technical Sciences, Head of  
the Chair (computer engineering) at  
the Penza Polytechnic Institute.

Is a prominent authority in the Soviet  
Union on computation and computer  
engineering, especially in the field of  
product quality control.

Has penned 200 publications, inclu-  
ding several study aids and a mono-  
graph.

MIR PUBLISHERS of Moscow publishes Soviet scientific and technical literature in twenty-five languages including all those most widely used. MIR translates texts into Russian, and from Russian originals produces books in following languages — English, German, French, Italian, Spanish, Portuguese, Czech, Slovak, Finnish, Hungarian, Mongolian, Arabic, Persian, Hindi, Tamil, Kannada, Vietnamese, Dari, Laotian, Cambodian, Greek, Bengali, Marathi, and Telugu. Titles include textbooks for higher technical and vocational schools, literature on the natural sciences and medicine (including textbooks for medical schools), popular science and science fiction. The contributors to MIR PUBLISHERS' list are leading Soviet scientists and engineers from all fields of science and technology, among them more than forty Members and Corresponding Members of the USSR Academy of Sciences. Skilled translators provide a high standard of translation from the original Russian. Many of the titles already issued by MIR PUBLISHERS have been adopted as textbooks and manuals at educational establishments in France, Switzerland, Cuba, Syria, India, Brasil and many other countries.

MIR PUBLISHERS' books in foreign languages can be purchased or ordered through booksellers in your country dealing with V/O "Mezhdunarodnaya Kniga", the authorised exporters.